Face Recognition Vendor Test (FRVT) 2012

Still Face Image and Video Concept, Evaluation Plan and API

Version 0.6

Patrick Grother, George W. Quinn, and Mei Ngan

Image Group
Information Access Division
Information Technology Laboratory

National Institute of Standards and Technology
U.S. Department of Commerce

May 15, 2012

Status of this Document

This document is the second public version. Changes since the first draft, v0.5 are marked in fuchsia. Comments and questions should be submitted to frvt2012@nist.gov.

Timeline of the FRVT 2012 Evaluation

Feb to Aug 2013	NIST documentation and reports released.
Dec 14, 2012	Open submission period ends.
Oct 31, 2012	Second interim report card released to submitting participants.
Oct 15, 2012	Deadline for submission for inclusion of results in second interim report card.
Sept 7, 2012	First interim report card released to submitting participants.
Aug 28, 2012	Deadline for submission for inclusion of results in first interim report card.
July 25, 2012 to Dec 14, 2012	FRVT 2012 open submission period.
July 25, 2012	Open submission period begins.
June 15, 2012	Final evaluation plan.
June 8, 2012	Comments period closes on second draft of this document.
May 15, 2012	Second draft evaluation plan (revised version of this document) for public comment.
April 30, 2012	Request that participants give non-binding no-commitment indication of whether they will participate in the test.
	Comments period closes on first draft of this document.
April 17, 2012	Initial draft evaluation plan (this document) for public comment.

```
      March 2012              April 2012               May 2012
Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa
             1  2  3     1  2  3  4  5  6  7           1  2  3  4  5
 4  5  6  7  8  9 10     8  9 10 11 12 13 14     6  7  8  9 10 11 12
11 12 13 14 15 16 17    15 16 17 18 19 20 21    13 14 15 16 17 18 19
18 19 20 21 22 23 24    22 23 24 25 26 27 28    20 21 22 23 24 25 26
25 26 27 28 29 30 31    29 30                   27 28 29 30 31

      June 2012               July 2012
Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa
                1  2     1  2  3  4  5  6  7
 3  4  5  6  7  8  9     8  9 10 11 12 13 14
10 11 12 13 14 15 16    15 16 17 18 19 20 21
17 18 19 20 21 22 23    22 23 24 25 26 27 28
24 25 26 27 28 29 30    29 30 31

     August 2012            September 2012           October 2012
Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa
          1  2  3  4                       1        1  2  3  4  5  6
 5  6  7  8  9 10 11     2  3  4  5  6  7  8     7  8  9 10 11 12 13
12 13 14 15 16 17 18     9 10 11 12 13 14 15    14 15 16 17 18 19 20
19 20 21 22 23 24 25    16 17 18 19 20 21 22    21 22 23 24 25 26 27
26 27 28 29 30 31       23 24 25 26 27 28 29    28 29 30 31
                        30

    November 2012           December 2012
Su Mo Tu We Th Fr Sa    Su Mo Tu We Th Fr Sa
             1  2  3                       1
 4  5  6  7  8  9 10     2  3  4  5  6  7  8
11 12 13 14 15 16 17     9 10 11 12 13 14 15
18 19 20 21 22 23 24    16 17 18 19 20 21 22
25 26 27 28 29 30       23 24 25 26 27 28 29
                        30 31
```

Major Changes since MBE 2010

Please note that this document is derived from the MBE-STILL 2010 API document for continuity and to aid implementers of the FRVT 2012 API. Any new or updated changes since MBE are highlighted in green. Any editor's notes are highlighted in yellow.

– For this test, Windows machines will not be used. Windows-compiled libraries are not permitted. All software must run under Linux (see section 1.20).

– The FRVT 2012 API is written in the C++ language. Participants are required to provide their library in a format that is linkable using g++ (see 1.20).

– This evaluation contains new focus areas, which include:

- Age, gender, and expression neutrality estimation for still images (see section 1.8)
- Dedicated API for video data (see section 3.6)

- Reporting minimum cost recognition (see section 1.15)

– New datasets will be used for FRVT 2012 and will contain individuals spanning a full age range.

– The header/source files for the API will be made available to implementers at http://nigos.nist.gov:8080/frvt2012/.

Table of Contents

List of Figures

List of Tables

Acknowledgements

— The authors are grateful to the experts who made extensive comments on the first version of this document.

Project History

— May 15, 2012 – Release of revised API, v0.6.

— April 17, 2012 - Release of first public draft of the Face Recognition Vendor Test (FRVT) 2012 – Concept, Evaluation Plan and API v0.5.

— June 17, 2010 – Published public report of MBE-STILL 2010 test (NISTIR 7709 – Report on the Evaluation of 2D Still-Image Face Recognition Algorithms) linked from http://face.nist.gov/mbe.

— August 2009 - Briefed large scale 1:N proposal to U. S. Government sponsors

Terms and definitions

The abbreviations and acronyms of Table 1 are used in many parts of this document.

Table 1 – Abbreviations

FNIR	False negative identification rate
FPIR	False positive identification rate
FMR	False match rate
FNMR	False non-match rate
FRVT	NIST’s Face Recognition Vendor Test program
FTS	Failure to Search
FTX	Failure to extract features from an enrollment image
GFAR	Generalized false accept rate
GFRR	Generalized false reject rate
DET	Detection error tradeoff characteristic: For verification this is a plot of FNMR vs. FMR (sometimes as normal deviates, sometimes on log-scales). For identification this is a plot of FNIR vs. FPIR.
INCITS	InterNational Committee on Information Technology Standards
ISO/IEC 19794	ISO/IEC 19794-5: Information technology — Biometric data interchange formats — Part 5:Face image data. First edition: 2005-06-15. (See Bibliography entry).
MBE	NIST's Multiple Biometric Evaluation program
NIST	National Institute of Standards and Technology
SDK	The term Software Development Kit refers to any library software submitted to NIST. This is used synonymously with the terms "implementation" and "implementation under test".

1. FRVT

1.1. Scope

This document establishes a concept of operations and an application programming interface (API) for evaluation of face recognition implementations submitted to NIST's Face Recognition Vendor Test 2012. See

http://www.nist.gov/itl/iad/ig/frvt-2012.cfm for all FRVT 2012 documentation.

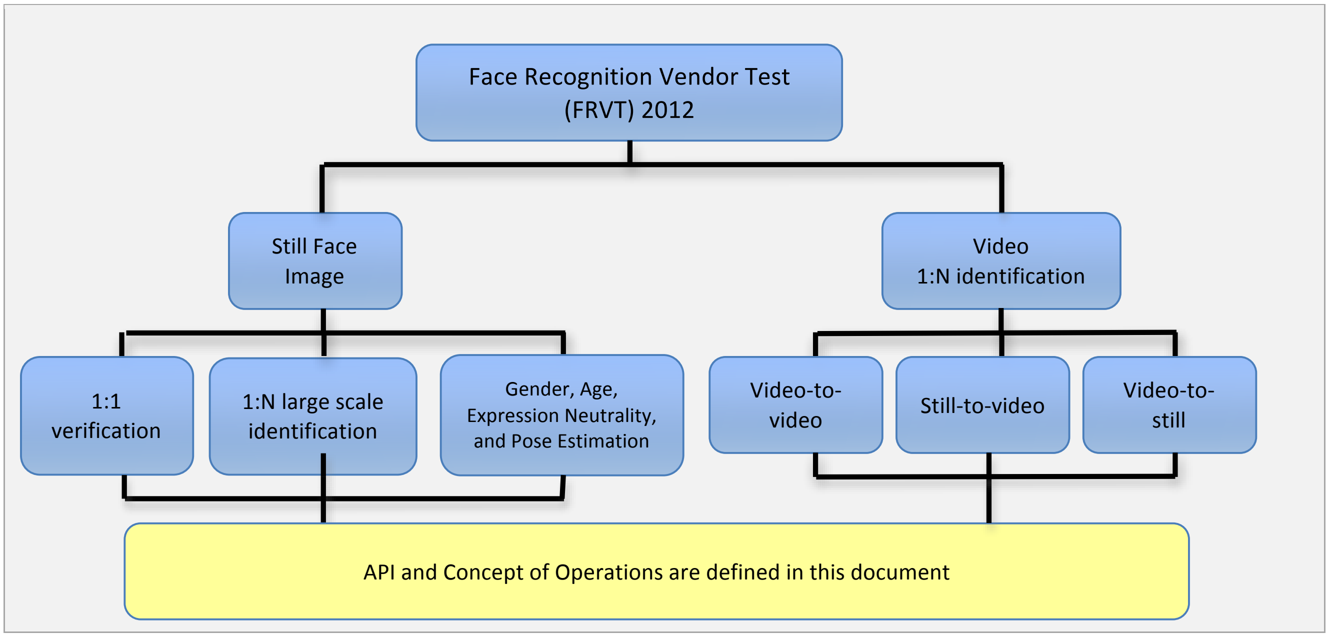

Figure 1 – Organization and documentation of the FRVT 2012

1.2. Audience

Universities and commercial entities with capabilities in any of the following areas are invited to participate in the FRVT 2012 Face test.

— Identity verification with face recognition algorithms.

— Large scale identification implementations.

— Profile view recognition.

— Organizations with a capability to assess age, gender, expression neutrality, and/or pose orientation of a face in an image.

— Face recognition in video capability

Organizations will need to implement the API defined in this document. Participation is open worldwide. There is no charge for participation. While NIST intends to evaluate technologies that could be readily made operational, the test is also open to experimental, prototype and other technologies.

1.3. Market drivers

This test is intended to support a plural marketplace of face recognition systems. While the dominant application, in terms of revenue, has been one-to-many search for driving licenses and visa issuance, the deployment of one-to-one face recognition has re-emerged with the advent of the e-Passport verification projects[1]. In addition, there remains considerable activity in the use of FR for surveillance applications.

These applications are differentiated by the population size (and other variables). In the driving license duplicate detection application, the enrollment database might exceed 10^7 people. In the surveillance application, the watchlist size can readily extend to 10^4.

1.4. Offline testing

While this set of tests is intended as much as possible to mimic operational reality, this remains an offline test executed on databases of images. The intent is to assess the core algorithmic capability of face recognition algorithms. This test will be conducted purely offline - it does not include a live human-presents-to-camera component. Offline testing is attractive because it allows uniform, fair, repeatable, and efficient evaluation of the underlying technologies. Testing of implementations under a fixed API allows for a detailed set of performance related parameters to be measured.

1.5. Phased testing

To support research and development efforts, this testing activity will embed multiple rounds of testing. These test rounds are intended to support improved performance. Once the test commences, NIST will test implementations on a first-come-first-served basis and will return results to providers as expeditiously as possible. Providers may submit revised SDKs to NIST only after NIST provides results for the prior SDK. The frequency with which a provider may submit SDKs to NIST will depend on the times needed for developer preparation, transmission to NIST, validation, execution and scoring at NIST, and developer review and decision processes.

For the number of SDKs that may be submitted to NIST see section 1.10.

1.6. Interim reports

The performance of each SDK will be reported in a "score-card". This will be provided to the participant. While the score cards may be used by the provider for arbitrary purposes, they are intended to facilitate development. The score cards will

– be machine generated (i.e. scripted),
– be provided to participants with identification of their implementation,
– include timing, accuracy and other performance results,
– include results from other implementations, but will not identify the other providers,
– be expanded and modified as revised implementations are tested, and as analyses are implemented,
– be generated and released asynchronously with SDK submissions,
– be produced independently of the other status of other providers' implementations,
– be regenerated on-the-fly, primarily whenever any implementation completes testing, or when new analysis is added.

NIST does not intend to release these test reports publicly. NIST may release such information to the U.S. Government test sponsors. While these reports are not intended to be made public, NIST can only request that agencies not release this content.

1.7. Final reports

NIST will publish one or more final public reports. NIST may also

– publish additional supplementary reports (typically as numbered NIST Interagency Reports),
– publish in other academic journals,

[1] These match images acquired from a person crossing a border against the ISO/IEC 19794-5 facial image stored on the embedded ISO/IEC 7816 + ISO/IEC ISO 14443 chips.

– present results at conferences and workshops (typically PowerPoint).

Our intention is that the final test reports will publish results for the best-performing implementation from each participant. Because “best” is ill-defined (accuracy vs. time vs. template size, for example), the published reports may include results for other implementations. The intention is to report results for the most capable implementations (see section 1.13, on metrics). Other results may be included (e.g. in appendices) to show, for example, examples of progress or tradeoffs. IMPORTANT: Results will be attributed to the providers.

1.8. Application scenarios

The test will include one-to-one verification tests and one-to-many identification tests[7] [MBE 2010, IREX III] for still images. It will also include one-to-many identification tests for video sequences. As described in Table 2, the test is intended to represent:

— Close-to-operational use of face recognition technologies in identification applications in which the enrolled dataset could contain images from up to three million persons.

— Verification scenarios in which still images are compared.

— Pose, age, gender, and expression neutrality estimation.

— Identification applications for face recognition in video

Table 2 – Subtests supported under the FRVT 2012 Still Image activity

#		A	B	C	D	V
1.	Aspect	1:1 verification	1:1 verification with enrollment database – Not Supported	1:N identification	Pose Conformance, Age, Gender, and Expression neutrality Estimation	Video-video, Still-video, video-still
2.	Enrollment dataset	None, application to single images	In MBE 2010, this class supported 1:1 verification with an enrollment database. This will not be supported for FRVT 2012.	N enrolled subjects	None, application to single images	N enrolled sequences
3.	Prior NIST test references	Equivalent to 1 to 1 matching in [MBE 2010]		Equivalent to 1 to N matching in [MBE 2010]		
4.	Example application	Verification of e-Passport facial image against a live border-crossing image.		Open-set identification of an image against a central database, e.g. a search of a mugshot against a database of known criminals.	During capture, algorithm assesses whether face is frontal or not, or estimates pose. Frontal pose is required in formal standards because non-frontal pose eventually degrades face recognition accuracy.	Open-set identification against a central database, e.g. a search of a wanted criminal through a live-video surveillance system at an airport who may attempt to flee the country
5.	Score or feature space normalization support	Vendor uses normalization techniques over SDK-internal datasets		Any score or feature based statistical normalization techniques-are applied against enrollment database		Any score or feature based statistical normalization techniques-are applied against enrollment database
6.	Intended number of subjects	Up to $O(10^5)$		Up to $O(10^7)$ but dependence on N will be computed. From $O(10^2)$ upwards.	Expected $O(10^3)$	Expected $O(10^3)$
7.	Number of images per individual	Variable, see section 1.11.		Variable, see section 1.11.	1	Variable

NOTE 1: The vast majority of images are color. The API supports both color and greyscale images.

NOTE 2: For the operational datasets, it is not known what processing was applied to the images before they were archived. So, for example, we do not know whether gamma correction was applied. NIST considers that best practice, standards and operational activity in the area of image preparation remains weak.

1.9. Image source labels

NIST may mix images from different source in an enrollment set. For example, NIST could combine N/2 mugshot images and N/2 visa images into a single enrollment dataset. For this reason, in the data structure defined in clause 2.3.3, each image is accompanied by a "label" which identifies the set-membership images. The legal values for labels are given in clause 2.3.2.

1.10. Options for participation

The following rules apply:

— A participant must properly follow, complete and submit the Annex A Participation Agreement. This must be done once. It is not necessary to do this for each submitted SDK.

— All participants shall submit at least one class A SDK, one class D SDK, or one class V SDK.

— A class A SDK shall be sent before, or concurrently with, any class C SDK.

— Class C, D, and V SDKs are optional. Those only interested in submitting a class D or class V SDK may do so without having to submit a class A SDK.

— Any SDK shall implement exactly one of the functionalities defined in Table 3. So, for example, the 1:1 functionality of a class A SDK shall not be merged with that of a class C SDK.

— NIST cannot conduct surveys over runtime parameters - NIST must limit the extent to which participants are able to train on the test data.

— Participants are asked to submit no more than two SDKs per class (ie. two class A, for class C – [two class CP, two class CN], two class D, two class V). We recommend participants review the submission schedule deadlines for inclusion in the first and second interim report cards at the beginning of this document. Please contact the FRVT 2012 POC if you intend on submitting more than two SDKs per class.

Table 3 – FRVT 2012 classes of participation

Function	1:1 verification	1:1 verification with enrollment database	1:N identification	Pose conformance, Age, Gender, and Expression neutrality estimation	Video
Class label	A	B	C [CP & CF, see Table 5]	D	V
Co-requisite class SDK	None	Not Supported	A	None	None
API requirements	3.1 + 3.2 + 3.3	Not Supported	3.1 + 3.2 + 3.4	3.1 + 3.5	3.6

Class A might be preferred by academic institutions because the API supports the elemental hypothesis test: "are the images from the same person or not?"

1.11. Use of multiple images per person

Some of the proposed datasets includes K > 2 images per person for some persons. This affords the possibility to model a recognition scenario in which a new image of a person is compared against all prior images[3]. Use of multiple images per person has been shown to elevate accuracy over a single image [FRVT2002b, MBE 2010].

For this test, NIST will enroll K ≥ 1 images under each identity. Normally the probe will consist of a single image, but NIST may examine the case that it could consist of multiple images. Ordinarily, the probe images will be captured after the enrolled images of a person[4]. The method by which the face recognition implementation exploits multiple images is not regulated: The test seeks to evaluate developer provided technology for multi-instance fusion. This departs from some prior NIST tests in which NIST executed fusion algorithms (e.g. [FRVT2002b]), and sum score fusion, for example, [MINEX]).

This document defines a template to be the result of applying feature extraction to a set of K ≥ 1 images. That is, a template contains the features extracted from one or more images, not generally just one. An SDK might internally fuse K feature sets into a single representation or maintain them separately - In any case the resulting proprietary template is contained in a contiguous block of data. All verification and identification functions operate on such multi-image templates.

The number of images per person will depend on the application area:

- In civil identity credentialing (e.g. passports, driving licenses) the images will be acquired approximately uniformly over time (e.g. five years for a Canadian passport). While the distribution of dates for such images of a person might be assumed uniform, a number of factors might undermine this assumption[5].
- In criminal applications the number of images would depend on the number of arrests[6]. The distribution of dates for arrest records for a person (i.e. the recidivism distribution) has been modeled using the exponential distribution, but is recognized to be more complicated. NIST currently estimates that the number of images will never exceed 100.

~~NIST will not use this API for video data.~~

1.12. Provision of photograph date information to the implementation

Due to face ageing effects, the utility of any particular enrollment image is dependent on the time elapsed between it and the probe image. In FRVT 2012, NIST intends to use the most recent image as the probe image, and to use one or more of the remaining prior images under a single enrolled identity.

1.13. Core accuracy metrics

Notionally the error rates for verification applications will be false match and false non-match error rates, FMR and FNMR.

For identification testing, the test will target open-universe applications such as benefits-fraud and watch-lists. It will not address the closed-set task because it is operationally uncommon.

While some one-to-many applications operate with purely rank-based metrics, this test will primarily target score-based identification metrics. Metrics are defined in Table 4. The analysis will survey over various rank and thresholds. Plots of the two error rates, parametric on threshold, will be the primary reporting mechanism.

[3] For example, if a banned driver applies for a driving license under a new name, and the local driving license authority maintains a driving license system in which all previous driving license photographs are enrolled, then the fraudulent application might be detected if the new image matched any of the prior images. This example implies one (elemental) method of using the image history.

[4] To mimic operational reality, NIST intends to maintain a causal relationship between probe and enrolled images. This means that the enrolled images of a person will be acquired before all the images that comprise a probe.

[5] For example, a person might skip applying for a passport for one cycle (letting it expire). In addition, a person might submit identical images (from the same photography session) to consecutive passport applications at five year intervals.

[6] A number of distributions have been considered to model recidivism, see ``Random parameter stochastic process models of criminal careers.'' In Blumstein, Cohen, Roth & Visher (Eds.), Criminal Careers and Career Criminals, Washington, D.C.: National Academy of Sciences Press, 1986.

Table 4 – Summary of accuracy metrics

Application	Metric		
1:1 Verification	FMR	=	Fraction of impostor comparisons that produce a similarity score greater than or equal to a threshold value
	FNMR	=	Fraction of genuine comparisons that produce a similarity score less than some threshold value
1:N Identification Primary identification metric.	FPIR	=	Fraction of searches that do not have an enrolled mate for which one or more candidate list entries is at or above a threshold
	FNIR	=	Fraction of searches that have an enrolled mate for which the mate is below a threshold
1:N Identification (with rank criteria) Secondary identification metric	FPIR	=	Fraction of searches that do not have an enrolled mate for which one or more candidate list entries is at or above a threshold
	FNIR	=	Fraction of searches that have an enrolled mate for which the mate is not in the best R ranks *and* at or above a threshold

NOTE: The metric on line B is a special case of the metric on line C: the rank condition is relaxed (R → N). Metric B is the primary metric of interest because the target application does not include a rank criterion.

FPIR will be estimated using probe images for which there is no enrolled mate.

NIST will extend the analysis in other areas, with other metrics, and in response to the experimental data and results.

1.14. Generalized accuracy metrics

Under the ISO/IEC 19795-1 biometric testing and reporting standard, a test must account for "failure to acquire" (FTA) and "failure to enroll" (FTE) events (e.g. elective refusal to make a template, or fatal errors). The way these are treated is application-dependent.

For verification, the appropriate metrics reported in FRVT 2012 will be generalized error rates (GFAR, GFRR). When single images are compared, (GFAR,GFRR) and (FMR,FNMR) will be equivalent if no failures are observed.

Similarly for identification, generalized error rates will be reported.

1.15. Reporting minimum cost recognition

This evaluation will investigate the use of cost parameters for application-specific algorithm optimization. The goal is to determine if matching algorithms can be modified to improve performance when the costs of errors are known in advance. The following cost model will be used as an evaluation metric for recognition performance:

$$E[Cost(\tau)] = (1-P_{Mated})FPIR(\tau)C_P + P_{Mated}\,FNIR(\tau)C_N$$

where P_{Mated} is the *a priori* probability that the user is mated, C_P is the cost of a false positive, C_N is the cost of a false negative, $FPIR(\tau)$ is the false positive identification rate, $FNIR(\tau)$ is the false negative identification rate, and τ is the operating threshold. The model estimates the expected cost per user attempt, which could be a measure of time, workload, money, etc. The participant is tasked with minimizing the cost for a predetermined and fixed set of cost parameters (C_P, C_N, and P_{Mated}).

Cost parameters are often chosen to correspond to a specific application. Consider a biometric system that provides bank vault access to specific individuals. One might reasonably set the cost of a false positive to be the monetary value of whatever is in the vault, and the cost of a false negative to a value that reflects the amount of inconvenience incurred from having to open the vault by some other method. Setting P_{Mated} to 0.1 assumes that one out of every ten access attempts is by an allowed user.

NIST requires each participant to submit two instances of the class C SDK, each corresponding to a different set of cost parameters. These parameters are defined in the table below. Class CP implementations penalize false positives heavily and false negatives lightly. Class CN implementations assign comparatively greater penalty to false negatives. For this class of implementations, suppression of false positives is less important.

Table 5 – Cost parameters for both submission types

Implementation Class	C_N	C_P	P_{Mated}
Class CP	1	10	0.01
Class CN	200	1	0.1

Additionally, failures to extract (FTXs) and failures to search (FTSs) will be treated differently depending on the implementation class. For Class CP implementations, both will be treated as failures in a positive recognition system (e.g. access control). This is the way NIST has handled FTXs and FTSs in prior evaluations. For Class CN implementations, FTXs and FTSs will be treated like failures in a negative recognition system (e.g. a watchlist). Failures in a negative recognition system increase the FPIR when they occur for non-mated searches, but do not increase the FNIR when they occur for mated searches. This differs from the way NIST has traditionally handled these types of failure.

The motivation for requiring participants to submit two implementations, each tuned to a different set of cost parameters, is to see if it is possible to change the shape of a DET to reduce cost for a specific set of cost parameters. Figure 2 plots standard DET curves for two identification algorithms. The two curves cross one another, making it impossible to state which is more accurate in any absolute sense. Since class C_FN implementations are penalized heavily for false negatives, and only lightly for false positives, both algorithms are expected to achieve their lowest cost toward the right end of the figure, where the blue curve performs better. Conversely, class C_FP implementations are penalized heavily for false positives but only lightly for false negatives. Thus, for this set of cost parameters, both algorithms are expected to achieve their lowest cost toward the left end of the figure, where the red curve performs better.

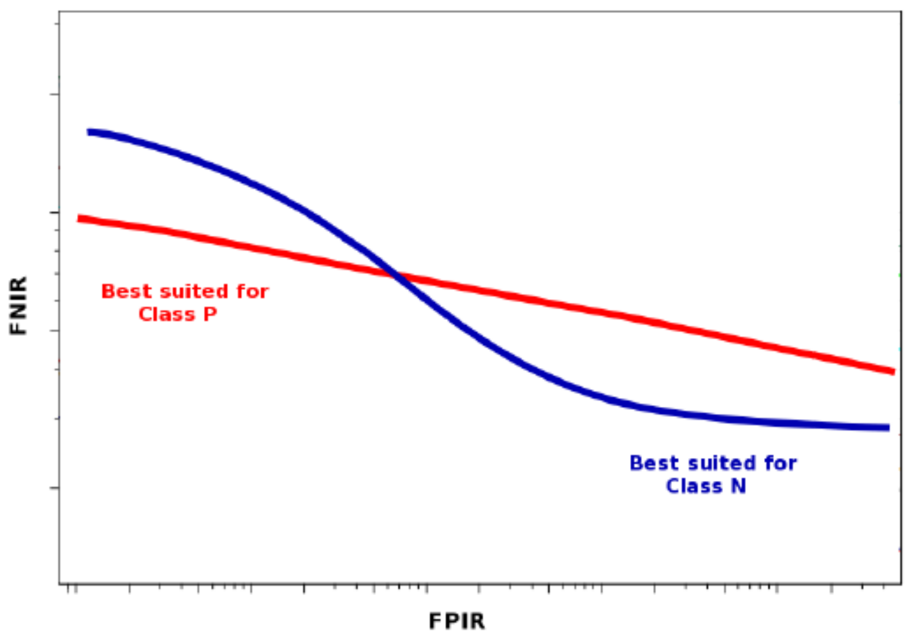

Figure 2 – Notional DET plots demonstrating how the two classes place greater emphasis on different regions of the DET

1.16. Reporting template size

Because template size is influential on storage requirements and computational efficiency, this API supports measurement of template size. NIST will report statistics on the actual sizes of templates produced by face recognition implementations submitted to FRVT 2012. NIST may report statistics on runtime memory usage. Template sizes were reported in the IREX III test[7] and the MBE-STILL 2010 test[8].

1.17. Reporting computational efficiency

As with other tests, NIST will compute and report recognition accuracy. In addition, NIST will also report timing statistics for all core functions of the submitted SDK implementations. This includes feature extraction, 1:1 and 1:N recognition, and pose conformance, age, gender, and expression neutrality estimation. For an example of how efficiency can be reported, see the final report of the IREX III test[7] and the MBE-STILL 2010 test[8].

Note that face recognition applications optimized for pipelined 1:N searches may not demonstrate their efficiency in pure 1:1 comparison applications.

1.18. Exploring the accuracy-speed trade-space

~~Organizations may enter two or more SDKs per class.~~ NIST will explore the accuracy vs. speed tradeoff for face recognition algorithms running on a fixed platform. NIST will report both accuracy and speed of the implementations tested. While NIST cannot force submission of "fast vs. slow" variants, participants may choose to submit variants on some other axis (e.g. "experimental vs. mature") implementations. NIST encourages “fast-less-accurate vs. slow-more-accurate” with a factor of three between the speed of the fast and slow versions.

1.19. Hardware specification

NIST intends to support high performance by specifying the runtime hardware beforehand. There are several types of computer blades that may be used in the testing. The blades are labeled as Dell M905, M910, M605, and M610. The following list gives some details about the hardware of each blade type:

- Dell M605 - Dual Intel Xeon E5405 2 GHz CPUs (4 cores each)
- Dell M610 - Dual Intel Xeon X5680 3.3 GHz CPUs (6 cores each)
- Dell M905 - Quad AMD Opteron 8376HE 2 GHz CPUs[9] (4 cores each)
- Dell M910 - Dual Intel Xeon X7560 2.3 GHz CPUs (8 cores each)

Each CPU has 512K cache. The bus runs at 667 Mhz. The main memory is 192 GB Memory as 24 8GB modules. We anticipate that 16 processes can be run without time slicing.

NIST is requiring use of 64 bit implementations throughout. This will support large memory allocation to support 1:N identification task with image counts in the millions. For still images, if all templates were to be held in memory, the 192GB capacity implies a limit of ~19KB per template, for a 10 million image enrollment. For video, given the data expectations and the occurrence of faces in the imagery, we anticipate the developers will have sufficient memory for video templates. Note that while the API allows read access of the disk during the 1:N search, the disk is, of course, relatively slow.

Some of the section 3 API calls allow the implementation to write persistent data to hard disk. The amount of data shall not exceed 200 kilobytes per enrolled image. NIST will respond to prospective participants' questions on the hardware, by amending this section.

1.20. Operating system, compilation, and linking environment

The operating system that the submitted implementations shall run on will be released as a downloadable file accessible from http://nigos.nist.gov:8080/nist , which will be the 64-bit version of CentOS 6.2 running Linux kernel 2.6.32-220.

For this test, Windows machines will not be used. Windows-compiled libraries are not permitted. All software must run under Linux.

---

[7] See the IREX III test report: NIST Interagency Report 7836, linked from http://iris.nist.gov/irex

[8] See the MBE-STILL 2010 test report, NIST Interagency Report 7709, linked from http://face.nist.gov/mbe

[9] cat /proc/cpuinfo returns fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm 3dnowext 3dnow constant_tsc nonstop_tsc pni cx16 popcnt lahf_lm cmp_legacy svm extapic cr8_legacy altmovcr8 abm sse4a misalignsse 3dnowprefetch osvw

NIST will link the provided library file(s) to our C++ language test drivers. Participants are required to provide their library in a format that is linkable using g++ version 4.4.6. The standard libraries are:

```
/usr/lib64/libstdc++.so.6.0.13        lib64/libc.so.6 -> libc-2.12.so      lib64/libm.so.6 -> libm-2.12.so
```

A typical link line might be

```
g++ -I. -Wall -m64 -o frvt12test  frvt12test.cpp  -L.  –lfrvt2012_Enron_A_07
```

The Standard C++ library should be used for development of the SDKs. The prototypes from the still image API portion of this document will be written to a file "frvt2012.h" which will be included via

```
#include <frvt2012.h>
```

The prototypes from the video API portion of this document will be written to a file "frvt2012Video.h" which will be included via

```
#include <frvt2012Video.h>
```

The header files will be made available to implementers at http://nigos.nist.gov:8080/frvt2012/.

NIST will handle all input of images via the JPEG and PNG libraries, sourced, respectively from http://www.ijg.org/ and see http://libpng.org.

All compilation and testing will be performed on x86 platforms. Thus, participants are strongly advised to verify library-level compatibility with g++ (on an equivalent platform) prior to submitting their software to NIST to avoid linkage problems later on (e.g. symbol name and calling convention mismatches, incorrect binary file formats, etc.).

Dependencies on external dynamic/shared libraries such as compiler-specific development environment libraries are discouraged. If absolutely necessary, external libraries must be provided to NIST upon prior approval by the Test Liaison.

1.21. Software and Documentation

1.21.1. SDK Library and Platform Requirements

Participants shall provide NIST with binary code only (i.e. no source code). Header files (".h") are allowed, but these shall not contain intellectual property of the company nor any material that is otherwise proprietary. It is preferred that the SDK be submitted in the form of a single static library file. However, dynamically linked shared library files are permitted.

The core library shall be named according to Table 6. Additional shared object library files may be submitted that support this "core" library file (i.e. the "core" library file may have dependencies implemented in these other libraries).

Intel Integrated Performance Primitives (IPP) libraries are permitted if they are delivered as a part of the developer-supplied library package. It is the provider's responsibility to establish proper licensing of all libraries. The use of IPP libraries shall not inhibit the SDK's ability to run on CPUs that do not support IPP. Please take note that some IPP functions are multithreaded and threaded implementations may complicate comparative timing.

Access to any GPUs is not permitted.

Table 6 – Implementation library filename convention

Form	libFRVT2012_provider_class_sequence.ending				
Underscore delimited parts of the filename	libFRVT2012	provider	class	sequence	ending
Description	First part of the name, required to be this.	Single word name of the main provider EXAMPLE: Acme	Function classes supported in Table 3. EXAMPLE: C	A two digit decimal identifier to start at 00 and increment by 1 every time any SDK is sent to NIST. EXAMPLE: 07	Either .so or .a
Example	libFRVT2012_Acme_C_07.a				

NIST will report the size of the supplied libraries.

1.21.2. Configuration and developer-defined data

The implementation under test may be supplied with configuration files and supporting data files. The total size of the SDK, that is all libraries, include files, data files and initialization files shall be less than or equal to 1 073 741 824 bytes = 1024^3 bytes.

NIST will report the size of the supplied configuration files.

1.21.3. Installation and Usage

The SDK must install easily (i.e. one installation step with no participant interaction required) to be tested, and shall be executable on any number of machines without requiring additional machine-specific license control procedures or activation.

The SDK shall be installable using simple file copy methods. It shall not require the use of a separate installation program.

The SDK shall neither implement nor enforce any usage controls or limits based on licenses, number of executions, presence of temporary files, etc. The SDKs shall remain operable until April 30 2013.

Hardware (e.g. USB) activation dongles are not acceptable.

1.21.4. Hard disk space

FRVT 2012 participants should inform NIST if their implementations require more than 100K of persistent storage, per enrolled image on average.

1.21.5. Documentation

Participants shall provide complete documentation of the SDK and detail any additional functionality or behavior beyond that specified here. The documentation must define all (non-zero) developer-defined error or warning return codes.

1.21.6. Modes of operation

Individual SDKs provided shall not include multiple “modes” of operation, or algorithm variations. No switches or options will be tolerated within one library. For example, the use of two different “coders” by a feature extractor must be split across two separate SDK libraries, and two separate submissions.

1.21.7. Watermarking of images

The SDK functions shall not watermark or otherwise steganographically mark up the images.

1.22. Runtime behavior

1.22.1. Interactive behavior

The SDK will be tested in non-interactive “batch” mode (i.e. without terminal support). Thus, the submitted library shall not use any interactive functions such as graphical user interface (GUI) calls, or any other calls which require terminal interaction e.g. reads from “standard input”.

1.22.2. Error codes and status messages

The SDK will be tested in non-interactive “batch” mode, without terminal support. Thus, the submitted library shall run quietly, i.e. it should not write messages to "standard error" and shall not write to “standard output”. An SDK may write debugging messages to a log file - the name of the file must be declared in documentation.

1.22.3. Exception Handling

The application should include error/exception handling so that in the case of a fatal error, the return code is still provided to the calling application.

1.22.4. External communication

Processes running on NIST hosts shall not side-effect the runtime environment in any manner, except for memory allocation and release. Implementations shall not write any data to external resource (e.g. server, file, connection, or other process), nor read from such. If detected, NIST will take appropriate steps, including but not limited to, cessation of evaluation of all implementations from the supplier, notification to the provider, and documentation of the activity in published reports.

1.22.5. Stateless behavior

All components in this test shall be stateless, except as noted. This applies to face detection, feature extraction and matching. Thus, all functions should give identical output, for a given input, independent of the runtime history. NIST will institute appropriate tests to detect stateful behavior. If detected, NIST will take appropriate steps, including but not limited to, cessation of evaluation of all implementations from the supplier, notification to the provider, and documentation of the activity in published reports.

1.23. Threaded computations

Table 7 shows the limits on the numbers of threads a face recognition implementation may use. In many cases threading is not permitted (i.e. T=1) because NIST will parallelize the test by dividing the workload across many cores and many machines. For the functions where we allow multi-threading, e.g. in the 1:N test, NIST requires the provider to disclose the maximum number of threads to us. If that number is T, NIST will run the largest integer number of processes, P, in parallel such that $TP \leq 16$.

Table 7 – Number of threads allowed for each application

<table>
<tr><td></td><td>A</td><td>C</td><td>D</td><td>V</td></tr>
<tr><td>Function</td><td>1:1 verification</td><td>1:N identification</td><td>Pose conformance, Age, Gender, Expression neutrality estimation</td><td>Video</td></tr>
<tr><td>Feature extraction</td><td>1</td><td>1</td><td rowspan="4">1</td><td>1</td></tr>
<tr><td>Verification</td><td>1</td><td>NA</td><td>NA</td></tr>
<tr><td>Finalize enrollment (before 1:1 or 1:N)</td><td>NA</td><td>1 ≤ T ≤ 16</td><td>1 ≤ T ≤ 16</td></tr>
<tr><td>Identification</td><td>NA</td><td>1 ≤ T ≤ 16</td><td>1 ≤ T ≤ 16</td></tr>
</table>

For comparative timing, the IREX III[7] test report estimated a factor by which the speed of threaded algorithms would be adjusted. Non-threaded implementations will eliminate the need for NIST to apply such techniques [IREX III].

NIST will not run an implementation from participant X and an implementation from participant Y on the same machine at the same time.

To expedite testing, for single-threaded libraries, NIST will run up to P = 16 processes concurrently. NIST's calling applications are single-threaded

1.24. Time limits

The elemental functions of the implementations shall execute under the time constraints of Table 8. These times limits apply to the function call invocations defined in section 3. Assuming the times are random variables, NIST cannot regulate the maximum value, so the time limits are 90-th percentiles. This means that 90% of all operations should take less than the identified duration.

The time limits apply per image. When K images of a person are present, the time limits shall be increased by a factor K.

Table 8 – Processing time limits in milliseconds

	A	C	D	V
Function	1:1 verification	1:N identification	Pose	Video

	without enrollment database		conformance, Age, Gender, and Expression neutrality estimation	
Feature extraction enrollment	1000 (1 core)	1000 (1 core)		5 * class C per video frame
Feature extraction for verification or identification	1000 (1 core)	1000 (1 core)		5 * class C per video frame
Verification	5 (1 core)	NA	500 (1 core)	NA
Identification of one search image against 1,000,000 single-image **MULTIFACE** records.	NA	10000 (16 cores) or 160000 (1 core)		NA

For video: if there are 100 video frames in a video sequence, then the feature extraction time limit for the video would be 5 * 1000 (class C time limit) * 100 frames, which means the time limit for each video frame would be 5000 milliseconds.

In addition the enrollment finalization procedure is subject to a time limit, as follows. For an enrollment of one million single-image **MULTIFACE**s, the total time shall be less than 7200 seconds. The implementation can use up to 16 cores. This limit includes disk IO time.

1.25. Test datasets

This section is under development. The data has, in some cases, been estimated from initial small partitions. The completion of this section depends on further work. The information is subject to change. We intend to update this section as fully as possible.

NIST is likely to use other datasets, in addition.

Table 9 – Main image corpora (others will be used)

	Laboratory	FRVT 2002+2006 / HCINT	New Dataset	Multiple Encounter Database (MEDS)
Collection, environment	See FRVT 2006 Report, Phillips et al. NIST IR 7408.	Visa application process	Visa application process	Law enforcement booking
Live scan, Paper		Live	Live	Live, few paper
Documentation		See NIST IR 6965 [FRVT2002]	New	See NIST Special Database 32 Volume 1 (MEDS-I) and Volume 2 (MEDS-II)[10].
Compression from [MBE 2010][11]		JPEG mean size 9467 bytes. See [FRVT2002b]	JPEG mean size 17 kilobytes	JPEG ~ 20:1
Maximum image size		300 x 252	300 x 252	Mixed, some are 640x480 others are 768x960, some are smaller.
Minimum image size		300 x 252	300 x 252	
Eye to eye distance		Median = 71 pixels	Median = 71 pixels	mean=156, sd=46
Frontal		Yes, well controlled		Moderately well controlled Profile images will be included and labeled as such.
Full frontal geometry		Yes, in most cases. Faces may have small background than ISO FF requires.	Yes, in most cases. Faces may have small background than ISO FF requires.	Mostly not. Varying amounts of the torso are visible.
Intended use	1:1	1:1 and 1:N		1:N

[10] NIST Special Database 32, Volume 1 and Volume 2 are available at: http://www.nist.gov/itl/iad/ig/sd32.cfm. MEDS-II is an update to MEDS-I and was published in February 2011.
[11] Compression effects were studied under MBE 2010 in NIST Interagency Report 7830, linked from http://face.nist.gov/mbe

Age	University population	18 years and above	0 years and above	18 years and above

1.26. Quality analysis

NIST will examine the effectiveness of quality scores in predicting recognition accuracy. A quality score is computed from an input record during feature extraction. The default method of analysis will be the error vs. reject analysis document in P. Grother and E. Tabassi, *Performance of biometric quality measures*, IEEE Trans. PAMI, 29:531–543, 2007.

The default use-case is that the enrollment image is assumed to be pristine (in conformance with the ISO standard, for example), and quality is being used *during* a verification or identification transaction to select the image most likely to match the reference image. The reference image is assumed to be unavailable for matching during the collection.

For reasons of operational realism, metadata, such as a date of birth, will not be provided to the quality computation.

Analyses other than for the default case may be conducted.

1.27. Ground truth integrity

Some of the test databases will be derived from operational systems. They may contain ground truth errors in which

— a single person is present under two different identifiers, or

— two persons are present under one identifier, or

— in which a face is not present in the image.

If these errors are detected, they will be removed. NIST will use aberrant scores (high impostor scores, low genuine scores) to detect such errors. This process will be imperfect, and residual errors are likely. For comparative testing, identical datasets will be used and the presence of errors should give an additive increment to all error rates. For very accurate implementations this will dominate the error rate. NIST intends to attach appropriate caveats to the accuracy results. For prediction of operational performance, the presence of errors gives incorrect estimates of performance.

2. Data structures supporting the API

2.1. Overview

This section describes separate APIs for the core face recognition applications described in section 1.8. All SDK's submitted to FRVT 2012 shall implement the functions required by the rules for participation listed before Table 3.

2.2. Requirement

FRVT 2012 participants shall submit an SDK which implements the relevant C++ prototyped interfaces of clause 3. C++ was chosen in order to make use of some object-oriented features.

2.3. File formats and data structures

2.3.1. Overview

In this face recognition test, an individual is represented by K ≥ 1 two-dimensional facial images, and by subject and image-specific metadata.

2.3.2. Dictionary of terms describing images

Images will be accompanied by one of the labels given in Table 10. Face recognition implementations submitted to FRVT 2012 should tolerate images of any category.

Table 10 – Labels describing types of images

	Label as C++ string	Primary test area	Meaning
1.	"unknown"		Either the label is unknown or unassigned.

2.	"laboratory frontal controlled"	1:1	Frontal with controlled illumination
3.	"laboratory frontal uncontrolled"	1:1	Any illumination
4.	"laboratory nonfrontal controlled"	1:1	NOTE: There is no hyphen "-"
5.	"laboratory nonfrontal uncontrolled"	1:1	Any illumination, pose is unknown and could be frontal
6.	"visa"	1:N	Either a member of the FRVT 2002/2006 HCINT corpus or one of similar properties.
7.	"mugshot"	1:N	Either a member of the Multi-encounter law enforcement database or one of similar properties. The image is nominally frontal - See NIST Special Database 32[10].
8.	"profile"	1:N	The image is a profile image taken from the multi-encounter law enforcement database.

NIST intends to use "profile" images in this evaluation.

2.3.3. Data structures for encapsulating multiple images

The standardized formats for facial images are the ISO/IEC 19794-5:2005 and the ANSI/NIST ITL 1-2007 type 10 record. The ISO record can store multiple images of an individual in a standalone binary file. In the ANSI/NIST realm, K images of an individual are usually represented as the concatenation of one Type 1 record + K Type 10 records. The result is usually stored as an EFT file.

An alternative method of representing K images of an individual is to define a structure containing an image filename and metadata fields. Each file contains a standardized image format, e.g. PNG (lossless) or JPEG (lossy).

Table 11 – Structure for a single face

Removed fields: `dob, mob, yob, day, month, year, sex, race, height, and weight`

	C++ code fragment	Remarks
1.	`typedef struct sface`	
2.	`{`	
3.	`uint16_t image_width;`	Number of pixels horizontally
4.	`uint16_t image_height;`	Number of pixels vertically
5.	`uint16_t image_depth;`	Number of bits per pixel. Legal values are 8 and 24.
6.	`uint8_t format;`	Flag indicating native format of the image as supplied to NIST 0x01 = JPEG (i.e. compressed data) 0x02 = PNG (i.e. never compressed data)
7.	`uint8_t *data;`	Pointer to raster scanned data. Either RGB color or intensity. If image_depth == 24 this points to 3WH bytes RGBRGBRGB... If image_depth == 8 this points to WH bytes `IIIIIII`
8.	`string description;`	Single description of the image. The allowed values for this string are given in Table 10.
9.		
10.	`} ONEFACE;`	

Table 12 – Structure for a set of images from a single person

Removed fields: `numfaces`

Please note the change from `struct` [MBE 2010] to `typedef` [FRVT 2012] for this data structure.

	C++ code fragment	Remarks
1.	`typedef std::vector<ONEFACE> MULTIFACE;`	Vector containing F pre-allocated face images of the same person. The number of items stored in the vector is accessible via the vector::size() function.

2.3.4. Data structure for eye coordinates

SDKs should return eye coordinates of each enrolled facial image. This function, while not necessary for a recognition test, will assist NIST in assuring the correctness of the test database. The primary mode of use will be for NIST to inspect images for which eye coordinates are not returned, or differ between developer SDKs.

The eye coordinates shall follow the placement semantics of the ISO/IEC 19794-5:2005 standard - the geometric midpoints of the endocanthion and exocanthion (see clause 5.6.4 of the ISO standard).

Sense: The label "left" refers to subject's left eye (and similarly for the right eye), such that xright < xleft.

Table 13 – Structure for a pair of eye coordinates

	C++ code fragment	Remarks
1.	`typedef struct ohos`	
2.	`{`	
	`bool failed;`	If the eye coordinates have been computed and assigned successfully, this value should be set to false, otherwise true.
3.	`int16_t xleft;`	X and Y coordinate of the center of the subject's left eye. Out-of-range values (e.g. x < 0 or x >= width) indicate the implementation believes the eye center is outside the image.
4.	`int16_t yleft;`	
5.	`int16_t xright;`	X and Y coordinate of the center of the subject's right eye. Out-of-range values (e.g. x < 0 or x >= width) indicate the implementation believes the eye center is outside the image.
6.	`int16_t yright;`	
7.	`} EYEPAIR;`	

2.3.5. Data type for similarity scores

Identification and verification functions shall return a measure of the similarity between the face data contained in the two templates. The datatype shall be an eight byte double precision real. The legal range is [0, DBL_MAX], where the DBL_MAX constant is larger than practically needed and defined in the <limits.h> include file. Larger values indicate more likelihood that the two samples are from the same person.

Providers are cautioned that algorithms that natively produce few unique values (e.g. integers on [0,127]) will be disadvantaged by the inability to set a threshold precisely, as might be required to attain a false match rate of exactly 0.0001, for example.

2.4. File structures for enrolled template collection

An SDK converts a **MULTIFACE** into a template, using, for example the "convert_**MULTIFACE**_to_enrollment_template" function of section 3.4.3. To support the class C identification functions of Table 3, NIST will concatenate enrollment templates into a single large file. This file is called the EDB (for enrollment database). The EDB is a simple binary concatenation of proprietary templates. There is no header. There are no delimiters. The EDB may extend to hundreds of gigabytes in length

This file will be accompanied by a manifest; this is an ASCII text file documenting the contents of the EDB. The manifest has the format shown as an example in Table 14. If the EDB contains N templates, the manifest will contain N lines. The fields are space (ASCII decimal 32) delimited. There are three fields, all containing numeric integers. Strictly speaking, the third column is redundant.

Table 14 – Enrollment dataset template manifest

Field name	Template ID	Template Length	Position of first byte in EDB
Datatype required	Unsigned decimal integer	Unsigned decimal integer	Unsigned decimal integer
Datatype length required	4 bytes	4 bytes	8 bytes
Example lines of a manifest file appear to the right. Lines 1, 2, 3 and N appear.	90201744	1024	0
	163232021	1536	1024
	7456433	512	2560
	...		
	183838	1024	307200000

The EDB scheme avoids the file system overhead associated with storing millions of individual files.

2.5. Data structure for result of an identification search

All identification searches shall return a candidate list of a NIST-specified length. The list shall be sorted with the most similar matching entries list first with lowest rank. The data structure shall be that of Table 15.

Table 15 – Structure for a candidate

	C++ code fragment	Remarks
1.	`typedef struct candidate`	
2.	`{`	
3.	`    bool failed;`	If the candidate computation failed, this value is set to true. If the candidate is valid it should be set to false.
4.	`    uint32_t template_id;`	The Template ID integer from the enrollment database manifest defined in clause 0.
5.	`    double similarity_score;`	Measure of similarity between the identification template and the enrolled candidate. Higher scores mean more likelihood that the samples are of the same person. An algorithm is free to assign any value to a candidate. The distribution of values will have an impact on the appearance of a plot of false-negative and false-positive identification rates.
6.	`    double probability;`	An estimate of the probability that the biometric data and candidate belong to different persons, i.e. the probability that a score this large would be observed given that the pair of images are from different people = P(SCORE | IMPOSTOR). This value shall be on [0:1]. This is one minus the integral of the expected impostor distribution from 0 to the similarity score, i.e. the expected false match rate.
7.	`} CANDIDATE;`	

3. API Specification

3.1. Implementation identifiers

All implementations shall support the self-identification function of Table 16. This function is required to support internal NIST book-keeping. The version numbers should be distinct between any versions, which offer different algorithmic functionality.

Table 16 – Implementation identifiers

Prototype	int32_t get_pid(	
	string &sdk_identifier,	A developer-assigned ID. This shall be different for each submitted SDK.
	string &email_address);	Output
Description	This function retrieves a point-of-contact email address from the implementation under test.	
Output Parameters	sdk_identifier	4-character version ID code as hexadecimal integer. This will be used to identify the SDK in the results reports. This value should be changed every time an SDK is submitted to NIST. The value is developer assigned - format is not regulated by NIST. EXAMPLE: "011A". The value cannot be the empty string.
	email_address	Point of contact email address. The value cannot be the empty string.
Return Value	0	Success
	Other	Vendor-defined failure

3.2. Maximum template size

All implementations shall report the maximum expected template sizes. These values will be used by the NIST test harnesses to pre-allocate template data. The values should apply to a single image. For a **MULTIFACE** containing K images, NIST will allocate K times the value returned. The function call is given in Table 17.

Table 17 – Implementation template size requirements

Prototype	int32_t get_max_template_sizes(	
	uint32_t &max_enrollment_template_size,	Output
	uint32_t &max_recognition_template_size)	Output
Description	This function retrieves the maximum template size needed by the feature extraction routines.	
Output	max_enrollment_template_size	The maximum possible size, in bytes, of the memory needed to store feature

Parameters		data from a single enrollment image.
	max_recognition_template_size	The maximum possible size, in bytes, of the memory needed to store feature data from a single verification or identification image.
Return Value	0	Success
	Other	Vendor-defined failure

3.3. 1:1 Verification

3.3.1. Overview

The 1:1 testing will proceed in three phases: preparation of enrollment templates; preparation of verification templates; and matching. These are detailed in Table 18.

Table 18 – Functional summary of the 1:1 application

Phase	#	Name	Description	Performance Metrics to be reported by NIST
Initialization	I1	Initialization	Function to allow implementation to read configuration data, if any.	None
Enrollment	E1	Serial enrollment	Given K ≥ 1 input images of an individual, the implementation will create a proprietary enrollment template. NIST will manage storage of these templates. NIST requires that these operations may be executed in a loop in a single process invocation, or as a sequence of independent process invocations, or a mixture of both.	Statistics of the time needed to produce a template. Statistics of template size. Rate of failure to produce a template and rate of erroneous function.
Verification	V1	Serial verification	Given K ≥ 1 input images of an individual, the implementation will create a proprietary verification template. NIST will manage storage of these templates. NIST requires that these operations may be executed in a loop in a single process invocation, or as a sequence of independent process invocations, or a mixture of both.	Statistics of the time needed to produce a template. Statistics of template size. Rate of failure to produce a template and rate of erroneous function.
Matching (i.e. comparison)	C1	Serial matching	Given one proprietary enrollment template and one proprietary verification template, compare these and produce a similarity score. NIST requires that these operations may be executed in a loop in a single process invocation, or as a sequence of independent process invocations, or a mixture of both.	Statistics of the time taken to compare two templates. Accuracy measures, primarily reported as DETs.

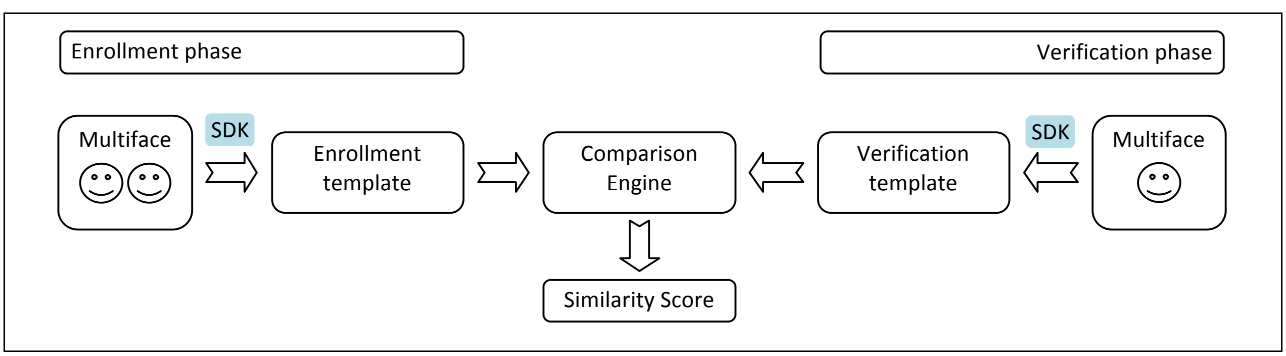

Figure 3 – Schematic of verification without enrollment database

3.3.2. API

Initialization of the implementation

Before any template generation or matching calls are made, the NIST test harness will make a call to the initialization of the function in Table 19.

Table 19 – SDK initialization

Removed fields: `num_descriptions`

Prototype	int32_t initialize_verification(		
	const string &configuration_location,		Input
	const std::vector<string> &descriptions);		Input
Description	This function initializes the SDK under test. It will be called by the NIST application before any call to the Table 20 functions convert_**MULTIFACE**_to_enrollment_template or convert_**MULTIFACE**_to_verification_template. The SDK under test should set all parameters.		
Input Parameters	configuration_location	A read-only directory containing any developer-supplied configuration parameters or run-time data files. The name of this directory is assigned by NIST. It is not hardwired by the provider. The names of the files in this directory are hardwired in the SDK and are unrestricted.	
	descriptions	A lexicon of labels one of which will be assigned to each image. EXAMPLE: The descriptions could be {"mugshot", "visa", "unknown"}. These labels are provided to the SDK so that it knows to expect images of these kinds. The number of items stored in the vector is accessible via the vector::size() function.	
Output Parameters	none		
Return Value	0	Success	
	2	Vendor provided configuration files are not readable in the indicated location.	
	8	The descriptions are unexpected, or unusable.	
	Other	Vendor-defined failure	

Template generation

The functions of Table 20 support role-specific generation of a template data. The format of the templates is entirely proprietary.

Table 20 – Template generation

Prototypes	int32_t convert_**MULTIFACE**_to_enrollment_template(		
	const MULTIFACE &input_faces,		Input
	uint32_t &template_size,		Output
	uint8_t *proprietary_template);		Output
	int32_t convert_**MULTIFACE**_to_verification_template(		
	const MULTIFACE &input_faces,		Input
	uint32_t &template_size,		Output
	uint8_t *proprietary_template,		Output
	uint8_t &quality);		Output
Description	This function takes a **MULTIFACE**, and outputs a proprietary template. The memory for the output template is allocated by the NIST test harness before the call i.e. the implementation shall not allocate memory for the result. In all cases, even when unable to extract features, the output shall be a template record that may be passed to the match_templates function without error. That is, this routine must internally encode "template creation failed" and the matcher must transparently handle this.		
Input Parameters	input_faces	An instance of a Table 12 structure. Implementations must alter their behavior according to the number of images contained in the structure.	
Output Parameters	template_size	The size, in bytes, of the output template	
	proprietary_template	The output template. The format is entirely unregulated. NIST will allocate a KT byte buffer for this template: The value K is the number of images in the **MULTIFACE**; the value T is output by the maximum template size functions of Table 17.	
	quality	An assessment of image quality. This is optional. The legal values are – [0,100] - The value should have a monotonic decreasing relationship with false non-match rate anticipated for this sample if it was compared with a pristine image of the same	

		person. So, a low value indicates high expected FNMR. – 255 - This value indicates a failed attempt to calculate a quality score. – 254 - This values indicates the value was not assigned.
Return Value	0	Success
	2	Elective refusal to process this kind of **MULTIFACE**
	4	Involuntary failure to extract features (e.g. could not find face in the input-image)
	6	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	8	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Other	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

Matching

Matching of one enrollment against one verification template shall be implemented by the function of Table 21.

Table 21 – Template matching

Prototype	int32_t match_templates(	
	const uint8_t *verification_template,	Input
	const uint32_t verification_template_size,	Input
	const uint8_t *enrollment_template,	Input
	const uint32_t enrollment_template_size,	Input
	double &similarity);	Output
Description	This function compares two opaque proprietary templates and outputs a similarity score, which need not satisfy the metric properties. NIST will allocate memory for this parameter before the call. When either or both of the input templates are the result of a failed template generation (see Table 20), the similarity score shall be -1 and the function return value shall be 2.	
Input Parameters	verification_template	A template from convert_**MULTIFACE**_to_verification_template().
	verification_template_size	The size, in bytes, of the input verification template $0 \le N \le 2^{32} - 1$
	enrollment_template	A template from convert_**MULTIFACE**_to_enrollment_template().
	enrollment_template_size	The size, in bytes, of the input enrollment template $0 \le N \le 2^{32} - 1$
Output Parameters	similarity	A similarity score resulting from comparison of the templates, on the range [0,DBL_MAX]. See section 2.3.5.
Return Value	0	Success
	2	Either or both of the input templates were result of failed feature extraction
	Other	Vendor-defined failure

3.4. 1:N Identification

3.4.1. Overview

The 1:N application proceeds in two phases, enrollment and identification. The identification phase includes separate pre-search feature extraction stage, and a search stage.

The design reflects the following *testing* objectives for 1:N implementations.

– support distributed enrollment on multiple machines, with multiple processes running in parallel
– allow recovery after a fatal exception, and measure the number of occurrences
– allow NIST to copy enrollment data onto many machines to support parallel testing
– respect the black-box nature of biometric templates
– extend complete freedom to the provider to use arbitrary algorithms
– support measurement of duration of core function calls
– support measurement of template size

647 **Table 22 – Procedural overview of the identification test**

Phase	#	Name	Description	Performance Metrics to be reported by NIST
Enrollment	E1	Initialization	Give the implementation advance notice of the number of individuals and images that will be enrolled. Give the implementation the name of a directory where any provider-supplied configuration data will have been placed by NIST. This location will otherwise be empty. The implementation is permitted **read-write-delete access** to the enrollment directory during this phase. The implementation is permitted read-only access to the configuration directory. After enrollment, NIST may rename and relocate the enrollment directory - the implementation should not depend on the name of the enrollment directory.	
	E2	Parallel Enrollment	For each of N individuals, pass multiple images of the individual to the implementation for conversion to a combined template. The implementation will return a template to the calling application. The implementation is permitted **read-only access** to the enrollment directory during this phase. NIST's calling application will be responsible for storing all templates as binary files. These will not be available to the implementation during this enrollment phase. Multiple instances of the calling application may run simultaneously or sequentially. These may be executing on different computers. The same person will not be enrolled twice.	Statistics of the times needed to enroll an individual. Statistics of the sizes of created templates. The incidence of failed template creations.
	E3	Finalization	Permanently finalize the enrollment directory. This supports, for example, adaptation of the image-processing functions, adaptation of the representation, writing of a manifest, indexing, and computation of statistical information over the enrollment dataset. The implementation is permitted **read-write-delete access** to the enrollment directory during this phase.	Size of the enrollment database as a function of population size N and the number of images. Duration of this operation. The time needed to execute this function shall be reported with the preceding enrollment times.
Pre-search	S1	Initialization	Tell the implementation the location of an enrollment directory. The implementation could look at the enrollment data. The implementation is permitted **read-only access** to the enrollment directory during this phase. Statistics of the time needed for this operation.	Statistics of the time needed for this operation.
	S2	Template preparation	For each probe, create a template from a set of input images. This operation will generally be conducted in a separate process invocation to step S2. The implementation is **permitted no access** to the enrollment directory during this phase. The result of this step is a search template.	Statistics of the time needed for this operation. Statistics of the size of the search template.
Search	S3	Initialization	Tell the implementation the location of an enrollment directory. The implementation should read all or some of the enrolled data into main memory, so that searches can commence. The implementation is permitted **read-only access** to the enrollment directory during this phase.	Statistics of the time needed for this operation.
	S4	Search	A template is searched against the enrollment database. The implementation is permitted **read-only access** to the enrollment directory during this phase.	Statistics of the time needed for this operation. Accuracy metrics - Type I + II error rates.

				Failure rates.

648 ### 3.4.2. Initialization of the enrollment session

649 Before any enrollment feature extraction calls are made, the NIST test harness will call the initialization function of Table
650 23.

651 **Table 23 – Enrollment initialization**

652 Removed fields: `num_descriptions`

Prototype	int32_t initialize_enrollment_session(	
	const string &configuration_location,	Input
	const string &enrollment_directory,	Input
	const uint32_t num_persons,	Input
	const uint32_t num_images,	Input
	const std::vector<string> &descriptions);	Input
Description	This function initializes the SDK under test and sets all needed parameters. This function will be called N=1 times by the NIST application immediately before any M ≥ 1 calls to convert_**MULTIFACE**_to_enrollment_template. The SDK should tolerate execution of P > 1 processes on the same machine each of which may be reading and writing to the enrollment directory. This function may be called P times and these may be running simultaneously and in parallel.	
Input Parameters	configuration_location	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollment_directory	The directory will be initially empty, but may have been initialized and populated by separate invocations of the enrollment process. When this function is called, the SDK may populate this folder in any manner it sees fit. Permissions will be read-write-delete.
	num_persons	The number of persons who will be enrolled $0 \le N \le 2^{32} - 1$ (e.g. 1million)
	num_images	The total number of images that will be enrolled, summed over all identities $0 \le M \le 2^{32} - 1$ (e.g. 1.8 million)
	descriptions	A lexicon of labels one of which will be assigned to each enrollment image. EXAMPLE: The descriptions could be {"mugshot", "visa"}. NOTE: The identification search images may or may not be labeled. An identification image may carry a label not in this set of labels. The number of items stored in the vector is accessible via the vector::size() function.
Output Parameters	none	
Return Value	0	Success
	2	The configuration data is missing, unreadable, or in an unexpected format.
	4	An operation on the enrollment directory failed (e.g. permission, space).
	6	The SDK cannot support the number of persons or images.
	8	The descriptions are unexpected, or unusable.
	Other	Vendor-defined failure

653 ### 3.4.3. Enrollment

654 A **MULTIFACE** is converted to a single enrollment template using the function of Table 24.

655 **Table 24 – Enrollment feature extraction**

Prototypes	int32_t convert_**MULTIFACE**_to_enrollment_template(	
	const MULTIFACE &input_faces,	Input
	std::vector<EYEPAIR> &output_eyes,	Output
	uint32_t &template_size,	Output
	uint8_t *proprietary_template);	Output
Description	This function takes a **MULTIFACE**, and outputs a proprietary template. The memory for the output template is allocated by the NIST test harness before the call i.e. the implementation shall not allocate memory for the result. If the function executes correctly (i.e. returns a zero exit status), the NIST calling application will store the template.	

	The NIST application will concatenate the templates and pass the result to the enrollment finalization function (see section 3.4.4). If the function gives a non-zero exit status: – If the exit status is 8, NIST will debug, otherwise – the test driver will ignore the output template (the template may have any size including zero) – the event will be counted as a failure to enroll. Such an event means that this person can never be identified correctly. IMPORTANT. NIST's application writes the template to disk. The implementation must not attempt writes to the enrollment directory (nor to other resources). Any data needed during subsequent searches should be included in the template, or created from the templates during the enrollment finalization function of section 3.4.4.	
Input Parameters	input_faces	An instance of a Table 12 structure. Implementations must alter their behavior according to the number of images contained in the structure.
Output Parameters	output_eyes	For each input image in the **MULTIFACE** the function shall return the estimated eye centers. The calling application will pre-allocate the correct number of EYEPAIR structures (i.e. one for each image in the **MULTIFACE**).
	template_size	The size, in bytes, of the output template
	proprietary_template	The format is entirely unregulated. NIST will allocate a KT byte buffer for this template: The value K is the number of images in the **MULTIFACE**; the value T is output by the maximum enrollment template size function of Table 17.
Return Value	0	Success
	2	Elective refusal to process this kind of **MULTIFACE**
	4	Involuntary failure to extract features (e.g. could not find face in the input-image)
	6	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	8	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Other	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.4.4. Finalize enrollment

After all templates have been created, the function of Table 25 will be called. This freezes the enrollment data. After this call the enrollment dataset will be forever read-only. This API does not support interleaved enrollment and search phases.

The function allows the implementation to conduct, for example, statistical processing of the feature data, indexing and data re-organization. The function may alter the file structure. It may increase or decrease the size of the stored data. No output is expected from this function, except a return code.

Table 25 – Enrollment finalization

Prototypes	int32_t finalize_enrollment (	
	const string &enrollment_directory,	Input
	const string &edb_name,	Input
	const string &edb_manifest_name);	Input
Description	This function takes the name of the top-level directory where enrollment database (EDB) and its manifest have been stored. These are described in section 0. The enrollment directory permissions will be read + write. The function supports post-enrollment developer-optional book-keeping operations and statistical processing. The function will generally be called in a separate process after all the enrollment processes are complete. This function should be tolerant of being called two or more times. Second and third invocations should probably do nothing.	
Input Parameters	enrollment_directory	The top-level directory in which enrollment data was placed. This variable allows an implementation to locate any private initialization data it elected to place in the directory.
	edb_name	The name of a single file containing concatenated templates, i.e. the EDB of section 0.

		While the file will have read-write-delete permission, the SDK should only alter the file if it preserves the necessary content, in other files for example. The file may be opened directly. It is not necessary to prepend a directory name.
	edb_manifest_name	The name of a single file containing the EDB manifest of section 0. The file may be opened directly. It is not necessary to prepend a directory name.
Output Parameters	None	
Return Value	0	Success
	2	Cannot locate the input data - the input files or names seem incorrect.
	4	An operation on the enrollment directory failed (e.g. permission, space).
	6	One or more template files are in an incorrect format.
	Other	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.4.5. Pre-search feature extraction

3.4.5.1. Initialization

Before **MULTIFACE**s are sent to the identification feature extraction function, the test harness will call the initialization function in Table 26.

Table 26 – Identification feature extraction initialization

Prototype	int32_t initialize_feature_extraction_session(	
	const string &configuration_location,	Input
	const string &enrollment_directory);	Input
Description	This function initializes the SDK under test and sets all needed parameters. This function will be called once by the NIST application immediately before any M ≥ 1 calls to convert_**MULTIFACE**_to_identification_template. The SDK should tolerate execution of P => 1 processes on the same machine each of which can read the configuration directory. This function may be called P times and these may be running simultaneously and in parallel. The implementation has read-only access to its prior enrollment data.	
Input Parameters	configuration_location	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollment_directory	The top-level directory in which enrollment data was placed and then finalized by the implementation. The implementation can parameterize subsequent template production on the basis of the enrolled dataset.
Output Parameters	none	
Return Value	0	Success
	2	The configuration data is missing, unreadable, or in an unexpected format.
	4	An operation on the enrollment directory failed (e.g. permission).
	Other	Vendor-defined failure

3.4.5.2. Feature extraction

A **MULTIFACE** is converted to an atomic identification template using the function of Table 27. The result may be stored by NIST, or used immediately. The SDK shall not attempt to store any data.

Table 27 – Identification feature extraction

Prototypes	int32_t convert_**MULTIFACE**_to_identification_template(	
	const MULTIFACE &input_faces,	Input
	std::vector<EYEPAIR> &output_eyes,	Output
	uint32_t &template_size,	Output

	uint8_t *identification_template);	Output
Description	This function takes a **MULTIFACE**, and outputs a proprietary template. The memory for the output template is allocated by the NIST test harness before the call i.e. the implementation shall not allocate memory for the result. If the function executes correctly, it returns a zero exit status. The NIST calling application may commit the template to permanent storage, or may keep it only in memory (the developer implementation does not need to know). If the function returns a non-zero exit status, the output template will be not be used in subsequent search operations. The function shall not have access to the enrollment data, nor shall it attempt access.	
Input Parameters	input_faces	An instance of a Table 12 structure. Implementations must alter their behavior according to the number of images contained in the structure.
Output Parameters	output_eyes	For each input image in the **MULTIFACE** the function shall return the estimated eye centers. The calling application will pre-allocate the correct number of EYEPAIR structures (i.e. one for each image in the **MULTIFACE**).
	template_size	The size, in bytes, of the output template
	identification_template	The output template for a subsequent identification search. The format is entirely unregulated. NIST will allocate a KT byte buffer for this template: The value K is the number of images in the input **MULTIFACE**; the value T is output by the maximum enrollment template size function of Table 17.
Return Value	0	Success
	2	Elective refusal to process this kind of **MULTIFACE**
	4	Involuntary failure to extract features (e.g. could not find face in the input-image)
	6	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	8	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Other	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.4.6. Initialization

The function of Table 28 will be called once prior to one or more calls of the searching function of Table 29. The function might set static internal variables so that the enrollment database is available to the subsequent identification searches.

Table 28 – Identification initialization

Prototype	int32_t initialize_identification_session(	
	const string &configuration_location,	Input
	const string &enrollment_directory);	Input
Description	This function reads whatever content is present in the enrollment_directory, for example a manifest placed there by the finalize_enrollment function.	
Input Parameters	configuration_location	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollment_directory	The top-level directory in which enrollment data was placed.
Return Value	0	Success
	Other	Vendor-defined failure

3.4.7. Search

The function of Table 29 compares a proprietary identification template against the enrollment data and returns a candidate list.

Table 29 – Identification search

Prototype	int32_t identify_template(	
	const uint8_t *identification_template,	Input
	const uint32_t identification_template_size,	Input
	const uint32_t candidate_list_length,	Input
	std::vector<CANDIDATE> &candidate_list,	Output

	bool &decision);	Output
Description	This function searches a template against the enrollment set, and outputs a list of candidates. NIST will pre-allocate the vector with candidates before the call.	
Input Parameters	identification_template	A template from convert_**MULTIFACE**_to_identification_template() - If the value returned by that function was non-zero the contents of identification_template will not be used and this function (i.e. identify_template) will not be called.
	identification_template_size	The size, in bytes, of the input identification template $0 \le N \le 2^{32} - 1$
	candidate_list_length	The number of candidates the search should return
Output Parameters	candidate_list	A vector containing "candidate_list_length" objects of candidates. The datatype is defined in section 2.5. Each candidate shall be populated by the implementation. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.
	decision	A best guess at whether there is a mate within the enrollment database. If there was a mate found, this value should be set to true, Otherwise, false. Many such decisions allow a single point to be plotted alongside a DET
Return Value	0	Success
	2	The input template was defective.
	Other	Vendor-defined failure

NOTE: Ordinarily the calling application will set the input candidate list length to operationally typical values, say $0 \le L \le 200$, and L << N. However, there is interest in the presence of mates much further down the candidate list. We may therefore extend the candidate list length such that L approaches N.

3.5. Pose conformance, age, gender, and expression neutrality estimation

The Multiple Encounter Database (MEDS)[10] contains facial images with varying age and gender (pose varies somewhat also but is not controlled). There are publicly available datasets that contain varying facial expression[12].

3.5.1. Pose conformance

The functions of this section support testing whether a face in an image has frontal pose. This supports conformance testing of, for example, the Full Frontal specification of the ISO standard [ISO]. The goal is to support a marketplace of products for acquisition time assessment of pose. This is important because pose is arguably the most influential covariate on face recognition error rates, and is not generally controllable by design of the acquisition system. This problem has been investigated in literature[13].

NIST encourages participants in this study to implement real-time video rate implementations, and also slower more accurate methods.

The functional specification here supports a DET analysis in which false-rejection of actually frontal images can be traded off against false acceptance of non-frontal images via a frontal-conformance parameter, t. The exact meaning of the "frontality" value returned by this function is not regulated by the NIST specification. However a reasonable implementation would embed a monotonic relationship between the output value and non-frontal angle (i.e. compound rotation involving azimuthal head yaw and pitch).

The formal ISO requirement is for five degree rotation in pitch and yaw. While the ISO standard establishes an eight degree limit on roll angle, this is of less importance. NIST will not consider roll angle.

3.5.2. Age

The functions of this section support estimation of the age of a face in an image. The process of age determination has potential application in at least the following areas:

[12] For example, the CMU Multi-PIE Face Database – http://www.multipie.org/ and others

[13] Erik Murphy-Chutorian and Mohan Manubhai Trivedi, "Head Pose Estimation in Computer Vision: A Survey," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol 31, no. 4, pp. 607-626, 2009.

- Age-based access control
- Age adaptive human machine interaction (e.g. marketing)
- Age invariant person identification
- Data mining and organization

Age estimation[14] has its own set of unique challenges when compared to other face image interpretation tasks, including limited inter-age group variation especially when dealing with mature subjects, diversity of aging variation between races and gender, and dependence on external factors such as health conditions, lifestyle, cosmetic surgery, etc.

3.5.3. Gender

The functions of this section support estimation of the gender[15] of a face in an image. Similar to age, gender is viewed as a soft biometric trait that has applications in surveillance, human-computer interaction and image retrieval systems. Gender could potentially be leveraged to index biometric databases and enhance the recognition accuracy of primary traits such as face.

3.5.4. Expression Neutrality

Facial expression recognition is an important aspect in interpersonal communication and human-machine interaction, having applications, for example, in building intelligent and more intuitive human-machine interfaces. ISO/IEC 19794-5:2005 establishes codes for facial expression. Clause 5.5.7 of that standard defines a neutral expression as "(non-smiling) with both eyes open and mouth closed".

EDITOR'S NOTE: Should NIST instead ask for implementations that can detect expressions according to Table 7 of the standard, which assigns integer codes to approximately these: "Neutral", "Smile closed", "Smile open", "Raised eyebrows", "Eyes looking away", "Squinting", "Frowning"?

3.5.5. API

Vendors may submit a class D SDK to evaluate performance on estimation of pose conformance, age, gender, and/or expression neutrality. The SDK must define a C++ class named exactly SdkEstimator, which subclasses from the Estimator class (see Table 30). At a minimum, the developer's SdkEstimator class must override at least one of the estimation functions and its corresponding initialization function from Table 30. To support those who only want to implement a subset of the class D estimation functions, any functions that are not overridden by the developer's SDK will default to the behavior specified in the "Base" Estimator Class (ie. return a value indicating function is "not implemented").

Table 30 – "Base" Estimator Class Structure

	C++ code fragment	Remarks
1.	`#include <vector>` `#include <string>`	
2.	`class Estimator {`	
3.	`public:`	
4.	`virtual ~Estimator();`	
5.	`virtual int32_t initialize_frontal_pose_estimation(` `const std::string &configuration_location);`	Pose conformance estimation initialization

[14] Xin Geng, Zhi-Hua Zhou, and Kate Smith-Miles, "Automatic Age Estimation Based on Facial Aging Patterns," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol 29, no. 12, pp. 2234-2240, 2007.

[15] C.H. Ting, U.U. Sheikh, and S.A.R. Abu-Bakar, "Gender estimation based on physiological features of the face", 10th International Conference on Information Science, ISSPA, pp. 201-204, 2010.

6.	`virtual int32_t estimate_frontal_pose_conformance(`
	`const ONEFACE &input_face,`
	`double &non_frontality);`
		Pose conformance estimation

7.	`virtual int32_t initialize_age_estimation(` `const std::string &configuration_location);`	Age estimation initialization

8.	`virtual int32_t estimate_age(`
	`const ONEFACE &input_face,`
	`int32_t &age);`
		Age (in years) estimation

9.	`virtual int32_t initialize_gender_estimation(` `const std::string &configuration_location);`	Gender estimation initialization

10.	`virtual int32_t estimate_gender(`
	`const ONEFACE &input_face,`
	`int8_t &gender,`
	`double &mf);`
		Gender estimation

~~11.~~	~~virtual int32_t initialize_age_and_gender_estimation(~~ ~~const std::string &configuration_location);~~	~~Combined age and gender estimation initialization~~
~~12.~~	~~virtual int32_t estimate_age_and_gender(~~ ~~const ONEFACE &input_face,~~ ~~int32_t &age,~~ ~~int8_t &gender,~~ ~~double &mf);~~	~~Combined age and gender estimation~~
13.	`virtual int32_t initialize_expression_estimation(` `const std::string &configuration_location);`	Expression neutrality estimation initialization

14.	`virtual int32_t estimate_expression_neutrality(`
	`const ONEFACE &input_face,`
	`double &expression_neutrality);`
		Expression neutrality estimation

15.	`};`	

An example of how the SdkEstimator class could be implemented is provided in Table 31 and Table 32. In the example, the pose estimation function and its corresponding initialization function is implemented. In this case, during runtime, the developer implementation of pose estimation will be evaluated. The rest of the unimplemented functions will default to the behavior specified in the "Base" Estimator class (see Table 30).

Table 31 – Example of SdkEstimator Class Declaration

	C++ code fragment – sdkestimator.h	Remarks
1.	`#include <frvt2012.h>`	
2.	`class SdkEstimator : public Estimator {`	
3.	`public:`	
4.	`SdkEstimator();`	Default constructor
5.	`~SdkEstimator();`	Default destructor
6.	`int32_t initialize_frontal_pose_estimation(` `const std::string &configuration_location);`	Pose conformance estimation initialization

7.	`int32_t estimate_frontal_pose_conformance(`
	`const ONEFACE &input_face,`
	`double &non_frontality);`
		Pose conformance estimation

8.	`};`	

Table 32 – Example of SdkEstimator Class Definition

	C++ code fragment – sdkestimator.cpp	Remarks
1.	`#include <sdkestimator.h>`	
2.	`SdkEstimator::SdkEstimator() { }`	Default constructor
3.	`SdkEstimator::~SdkEstimator() { }`	Default destructor

4.	`int32_t`
	`SdkEstimator::initialize_frontal_pose_estimation(`
	`     const std::string &configuration_location)`
	`{`
	`     return 0;`
	`}`
		Override the pose conformance estimation initialization function

5.	`int32_t`
	`SdkEstimator::estimate_frontal_pose_conformance(`
	`    const ONEFACE &input_face,`
	`    double &non_frontality)`
	`{`
	`    non_frontality = 0.1;`
	`    return 0;`
	`}`
		Override the pose conformance estimation function

6.	`};`	

The initialization functions of Table 33 will be called before one or more calls to the corresponding pose conformance, age, gender, and expression neutrality estimation functions. In other words, initialize_frontal_pose_estimation() will be called prior to estimate_frontal_pose_conformance(), initialize_age_estimation() will be called prior to estimate_age(), initialize_gender_estimation() will be called prior to estimate_gender(), ~~initialize_age_and_gender_estimation() will be called prior to estimate_age_and_gender()~~, and initialize_expression_estimation() will be called prior to estimate_expression_neutrality().

Table 33 – Initialization of Pose conformance, Age, Gender, and Expression neutrality estimation

Prototypes	int32_t initialize_frontal_pose_estimation(	
	const string &configuration_location);	Input
	int32_t initialize_age_estimation(	
	const string &configuration_location);	Input
	int32_t initialize_gender_estimation(	
	const string &configuration_location);	Input
	~~int32_t initialize_age_and_gender_estimation(~~	~~Combined age and gender estimation. This function may be implemented to improve timing performance with generating both estimates from within the same function.~~
	~~const string &configuration_location);~~	~~Input~~
	int32_t initialize_expression_estimation(	
	const string &configuration_location);	Input
Description	This function initializes the SDK under test. It will be called by the NIST application before any corresponding call to the Table 34 functions. The SDK under test should set all parameters.	
Input Parameters	configuration_location	A read-only directory containing any developer-supplied configuration parameters or run-time data files. The name of this directory is assigned by NIST. It is not hardwired by the provider. The names of the files in this directory are hardwired in the SDK and are unrestricted.
Output Parameters	none	
Return Value	0	Success
	2	Vendor provided configuration files are not readable in the indicated location.
	Other	Vendor-defined failure

Table 34 provides more details on the functions for computing a pose conformance, age, gender, and expression neutrality from an image.

Table 34 – Pose conformance, Age, Gender, Expression neutrality estimation

	int32_t estimate_frontal_pose_conformance(	Input
	const ONEFACE &input_face,	Output

	double &non_frontality);	
Prototypes	int32_t estimate_age(	
	const ONEFACE &input_face,	Input
	int32_t &age);	Output
	int32_t estimate_gender(	
	const ONEFACE &input_face,	Input
	int8_t &gender	Output
	double &mf);	Output
	~~int32_t estimate_age_and_gender(~~	
	~~const ONEFACE &input_face,~~	~~Input~~
	~~int32_t &age,~~	~~Output~~
	~~int8_t &gender,~~	~~Output~~
	~~double &mf);~~	~~Output~~
	int32_t estimate_expression_neutrality(	
	const ONEFACE &input_face,	Input
	double &expression_neutrality);	Output
Descriptions	estimate_frontal_pose_conformance - this function takes a **ONEFACE**, and outputs a non-frontality value for the image. The non-frontality value should increase with larger deviations from frontal pose. estimate_age – this function takes a **ONEFACE**, and outputs an age value (in years) for the image. estimate_gender - this function takes a **ONEFACE**, and outputs a gender value and a maleness-femaleness value for the image. ~~estimate_age_and_gender - this function takes a **ONEFACE**, and outputs both an age (in years) and gender value and a maleness-femaleness value for the image.~~ estimate_expression_neutrality – this function takes a **ONEFACE**, and an expression neutrality value for the image.	
Input Parameters	input_face	An instance of a Table 11 structure.
Output Parameters	non-frontality	Indication of how far from frontal the head pose is. The value should be on the range [0,1].
	age	Indication of the age (in years) of the person. The value should be on the range [0,100].
	gender	Indication of the gender of the person. Valid values are 0: Male 1: Female -1: Unknown
	mf	A real-valued measure of maleness-femaleness value on [0,1]. A value of 0 indicates certainty that the subject is a male, and a value of 1 indicates certainty that the subject is a female.
	expression_neutrality	ISO/IEC 19794-5:2005 establishes codes for facial expression. Clause 5.5.7 of that standard defines a neutral expression as “(non-smiling) with both eyes open and mouth closed”. SDKs shall report a real-valued measure of expression neutrality on [0,1] with 0 denoting large deviation from neutral and 1 indicating a fully neutral expression.
Return Value	0	Success
	2	Elective refusal to process this kind of **MULTIFACE**
	4	Involuntary failure to extract features (e.g. could not find face in the input-image)
	8	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Other	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.6. Video

EDITOR's NOTE: This entire section is NEW.

EDITOR'S NOTE: The API here is almost completely self-contained - it does not use any API data structures or functions from the still image section EXCEPT for the MULTIFACE structure for encapsulating still images.

3.6.1. Definitions

As shown in Table 35, the video API supports 1:N identification of video-to-video, video-to-still image, and still image-to-video. The following hold:

- A still image is a picture of one and only one person. One or more such images are presented to the implementation using a **MULTIFACE** data structure
- A video is a sequence of F ≥ 1 frames containing P ≥ 0 persons.
- A frame is 2D still image containing P ≥ 0 persons
- Any person might be present in 0 ≤ f ≤ F frames, and their presence may be non-contiguous (e.g. due to occlusion)
- Different videos contain different numbers of frames and people.
- A **ONEVIDEO** container is used to represent a video. It contains a small header and pointers to F frames.
- Any person found in a video is represented by proprietary template (feature) data contained with a **PERSONREP** data structure. A proprietary template contains information from one or more frames. Internally, it might embed multiple traditional still-image templates, or it might integrate feature data by tracking a person across multiple frames.
- A **PERSONREP** structure additionally contains a trajectory indicating the location of the person in each frame.

Please note that all of the code for the classes needed to implement the video API will be provided to implementers at http://nigos.nist.gov:8080/frvt2012/. A sample video will also be made available at the same link. The sample video is only approximately representative of the scene and is not an extraction from the actual video data that will be used in the evaluation. It is only intended to illustrate similarities in terms of camera placement relative to the subject and people behavior. It is not intended to represent the optical properties of the actual imaging systems, particularly the spatial sampling rate, nor the compression characteristics. More information will be released moving forward.

In the videos, the scenes capture people walking towards the camera. Sometimes, there are people walking in various transverse directions (ie. not parallel with the optical axis). The cameras have varying pitch angles, and the depth of scene varies between the cameras such that the sizes of the faces vary, with the following:

- Eye-to-eye distances range from approximately 40 pixels to 120 pixels
- Amount of time a face is fully visible in a scene can vary from approximately 0 to 5 seconds

Table 35 – API implementation requirements for Video

Function	Video-to-video	Still-to-video	Video-to-still
Enroll	Videos	Videos	Stills
Enrollment input datatype	**ONEVIDEO**	**ONEVIDEO**	**MULTIFACE**
Enrollment datatype	**PERSONREP**	**PERSONREP**	**PERSONREP**
Search	Video	Still	Video
Search input datatype	**ONEVIDEO**	**MULTIFACE**	**ONEVIDEO**
Search datatype	**PERSONREP**	**PERSONREP**	**PERSONREP**
Search result	**CANDIDATELIST**	**CANDIDATELIST**	**CANDIDATELIST**
API requirements	3.6.9 + 3.6.10 + 3.6.12 + 3.6.14	3.6.9 + 3.6.10 + 3.6.20 + 3.6.14	3.6.16 + 3.6.18 + 3.6.12 + 3.6.21

3.6.1.1. Video-to-video

Video-to-video identification is the process of enrolling N videos and then searching the enrollment database with a search video. During identification, the SDK shall return a set of indices of candidate videos that contain people who appear in the search video.

- N templates will be generated from 1 < M ≤ N enrollment videos.

– The N templates will be concatenated and finalized into a proprietary enrollment data structure.
– A **ONEVIDEO** will be converted to S ≥ 0 identification template(s) based on the number of people detected in the video.
– Each identification template generated will be searched against the enrollment database of templates generated from N still images.

3.6.1.2. Still image-to-video

Still image-to-video identification is the process of enrolling N videos and then searching the enrollment database with a template produced from a **MULTIFACE** as follows:

– N templates will be generated from 1 < M ≤ N enrollment videos.
– The N templates will be concatenated and finalized into a proprietary enrollment data structure.
– A **MULTIFACE** (still image) will be converted to an identification template.
– The identification template will be searched against the enrollment database of N templates.

3.6.1.3. Video-to-still image

Video-to-still image identification is the process of enrolling N **MULTIFACEs** (see Table 12) and then searching the enrollment database with templates from persons found in a video as follows

– N templates will be generated from N still-image **MULTIFACE**s.
– The N templates will be concatenated and finalized into a proprietary enrollment data structure.
– A **ONEVIDEO** will be converted to S identification template(s) based on the number of people detected in the video.
– Each of the S identification templates will be searched separately against the enrollment database of N templates.

3.6.2. Class for encapsulating a video sequence

Table 36 – ONEVIDEO Class

	C++ code fragment	Remarks
1.	`class ONEVIDEO`	
2.	`{` `private:`	
3.	`uint16_t frameWidth;`	Number of pixels horizontally of all frames
4.	`uint16_t frameHeight;`	Number of pixels vertically of all frames
5.	`uint8_t frameDepth;`	Number of bits per pixel for all frames. Legal values are 8 and 24.
6.	`uint16_t framesPerSec;`	The frame rate of the video sequence in seconds
7.	`public:` `std::vector<uint8_t*> data;`	Vector of pointers to data from each frame in the video sequence. The number of frames (ie. size of the vector) can be obtained by calling vector::size(). The i-th entry in data (ie. data[i]) points to frame_width x frame_height pixels of data for the i-th frame.
8.	`//Getter and Setter Methods`	
9.	`};`	

3.6.3. Class representing a pair of eye coordinates

Table 37 – EYEPAIR Class

	C++ code fragment	Remarks
1.	`class EYEPAIR`	
2.	`{` `private:`	
3.	`bool isSet;`	If the eye coordinates have been computed and assigned successfully, this value should be set to true, otherwise it should be set to false.
4.	`int16_t xLeft;` `int16_t yLeft;`	X and Y coordinate of the center of the subject's left eye. Out-of-range values (e.g. x < 0 or x >= width) indicate the implementation believes the eye center is outside the image.

5.	`int16_t xRight;` `int16_t yRight;`	X and Y coordinate of the center of the subject's right eye. Out-of-range values (e.g. x < 0 or x >= width) indicate the implementation believes the eye center is outside the image.
6.	`uint16_t frameNum`	For video: the frame number that corresponds to the video frame from where the eye coordinates was generated. (ie. the i-th frame from the video sequence). This field should not be set for eye coordinates for a single still image.
7.	`public:` `//getter/setter methods`	
8.	`};`	

3.6.4. Data type for representing a person's trajectory via eye coordinates from a video sequence

Table 38 – PersonTrajectory typedef

	C++ code fragment	Remarks
1.	`typedef std::vector<EYEPAIR> PersonTrajectory;`	Vector of **EYEPAIR** (see 3.6.3) objects for video frames where eyes were detected. This data structure should store eye coordinates for each video frame where eyes were detected for a particular person. For video frames where the person's eyes were not detected, the SDK shall not add an **EYEPAIR** to this data structure. If a face can be detected, but not the eyes, this structure could be populated with $(x,y)_{LEFT} == (x,y)_{RIGHT}$

3.6.5. Class for representing a person from a video sequence or an image

Table 39 – PERSONREP Class

	C++ code fragment	Remarks
1.	`class PERSONREP`	
2.	`{` `private:`	
3.	`PersonTrajectory eyeCoordinates;`	Data structure for capturing eye coordinates
4.	`PersonTemplate proprietaryTemplate;`	PersonTemplate is a wrapper to a uint8_t* for capturing proprietary template data representing a person from a video sequence or an image.
5.	`public:`	
6.	`PERSONREP(const uint64_t inSize);`	The constructor takes a size parameter and allocates memory of *inSize*. getPersonTemplatePtr() should be called to access the newly allocated memory for SDK manipulation. Please note that this class will take care of all memory allocation and de-allocation of its own memory. The SDK shall not de-allocate memory created by this class.
7.	`void pushBackEyeCoord(const EYEPAIR &eyes);`	This function should be used to add **EYEPAIR**s for the video frames or images where eye coordinates were detected.
8.	`uint8_t* getPersonTemplatePtr()` ~~const~~`;`	This function returns a uint8_t* to the template data.
9.	`uint64_t getPersonTemplateSize() const;`	This function returns the size of the template data.
10.	`//… getter methods, copy constructor,` `//… assignment operator`	
11.		
12.	`};`	

3.6.6. Class for result of an identification search

All identification searches shall return a candidate list of a NIST-specified length. The list shall be sorted with the most similar matching entries list first with lowest rank.

Table 40 – CANDIDATE Class

	C++ code fragment	Remarks
1.	`class CANDIDATE`	

2.	`{` `private:`	
3.	`bool isSet`	If the candidate is valid, this should be set to true. If the candidate computation failed, this should be set to false.
4.	`uint32_t templateId;`	The Template ID integer from the enrollment database manifest defined in clause 0.
5.	`double similarityScore;`	Measure of similarity between the identification template and the enrolled candidate. Higher scores mean more likelihood that the samples are of the same person. An algorithm is free to assign any value to a candidate. The distribution of values will have an impact on the appearance of a plot of false-negative and false-positive identification rates.
6.	`public:` `//getter/setter methods`	
7.	`};`	

3.6.7. Data type for representing a list of results of an identification search

Table 41 – CANDIDATELIST typedef

	C++ code fragment	Remarks
1.	`typedef std::vector<CANDIDATE> CANDIDATELIST;`	A vector containing objects of CANDIDATEs. The CANDIDATE class is defined in section 3.6.6.

3.6.8. Class representing return code values

Table 42 – ReturnCode class

	C++ code fragment	Remarks
	`class ReturnCode {` `public:`	
1.	`enum Status`	
2.	`{`	
3.	`Success=0,`	Success
4.	`MissingConfig=1,`	The configuration data is missing or unreadable
5.	`EnrollDirFailed=2,`	An operation on the enrollment directory failed
6.	`InitNumData=3,`	The SDK can't support the number of images or videos
7.	`InitBadDesc=4,`	The image descriptions are unexpected or unusable
8.	`RefuseInput=5,`	Elective refusal to process this kind of input (ONEVIDEO or MULTIFACE)
9.	`FailExtract=6,`	Involuntary failure to extract features
10.	`FailTempl=7,`	Elective refusal to produce a template
11.	`FailParse=8,`	Cannot parse input data
12.	`FinInputData=9,`	Cannot locate input data
13.	`FinTemplFormat=10,`	One or more template files are in an incorrect format
14.	`IdBadTempl=11,`	The input template was defective
15.	`Vendor=88`	Vendor-defined failure
16.	`};`	
17.	`ReturnCode(const Status inStatus);`	Constructor that takes an input parameter of a Status enum value. All of the functions that need to be implemented for the Video API return an instantiation of a ReturnCode object with a valid status value passed in as a parameter.
18.	`Status getStatus() const;`	Getter method to return status value
19.	`private:`	
20.	`Status status;`	Member variable for storing status
21.	`};`	

3.6.9. The VideoEnrollment Interface

The abstract class VideoEnrollment must be implemented by the SDK developer in a class named exactly SdkVideoEnrollment. The processing that takes place during each phase of the test is done via calls to the methods declared in the interface as pure virtual, and therefore is to be implemented by the SDK. The test driver will call these methods, handling all return values.

	C++ code fragment	Remarks
1.	`class VideoEnrollment`	
2.	`{` `public:`	
3.	`virtual ReturnCode getPid(` `string &sdkId, string &email) = 0;`	Return the sdk identifier and email

4.	`virtual ReturnCode initialize(`
	`const string &configDir,`
	`const string &enrollDir,`
	`const uint32_t numVideos) = 0 ;`
		Initialize the enrollment session.

5.	`virtual ReturnCode generateEnrollmentTemplate(`
	`const ONEVIDEO &inputVideo,`
	`vector<PERSONREP> &enrollTemplates) = 0;`
		Generate enrollment template(s) for the persons detected in the input video. This function takes an **ONEVIDEO** (see 3.6.2) as input and populates a vector of **PERSONREP** (see 3.6.5) with the number of persons detected from the video sequence. The implementation could call vector::push_back to insert into the vector.

6.	`// Destructor`	
7.	`};`	

3.6.9.1. Implementation identifier

Table 43 – VideoEnrollment::getPid

Prototype	ReturnCode getPid(	
	string &sdkId,	A developer-assigned ID. This shall be different for each submitted SDK.
	string &email);	Output
Description	This function retrieves a point-of-contact email address from the implementation under test.	
Output Parameters	sdkId	4-character version ID code as hexadecimal integer. This will be used to identify the SDK in the results reports. This value should be changed every time an SDK is submitted to NIST. The value is developer assigned - format is not regulated by NIST. EXAMPLE: "011A". The value cannot be the empty string.
	email	Point of contact email address. The value cannot be the empty string.
ReturnCode	Success	Success
	Vendor	Vendor-defined failure

3.6.9.2. Initialization of the video enrollment session

Before any enrollment feature extraction calls are made, the NIST test harness will call the initialization below for video-to-video and still image-to-video.

Table 44 – VideoEnrollment::initialize

Prototype	ReturnCode initialize(	
	const string &configDir,	Input
	const string &enrollDir,	Input
	const uint32_t numVideos);	Input
Description	This function initializes the SDK under test and sets all needed parameters. This function will be called N=1 times by the NIST application immediately before any M ≥ 1 calls to generateEnrollmentTemplate. The SDK should tolerate execution of P > 1 processes on the same machine each of which may be reading and writing to the	

	enrollment directory. This function may be called P times and these may be running simultaneously and in parallel.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The directory will be initially empty, but may have been initialized and populated by separate invocations of the enrollment process. When this function is called, the SDK may populate this folder in any manner it sees fit. Permissions will be read-write-delete.
	numVideos	The total number of videos that will be passed to the SDK for enrollment.
Output Parameters	none	
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission, space).
	InitNumData	The SDK cannot support the number of videos.
	Vendor	Vendor-defined failure

3.6.9.3. Video enrollment

An **ONEVIDEO** is converted to enrollment template(s) for each person detected in the **ONEVIDEO** using the function below.

Table 45 – VideoEnrollment::generateEnrollmentTemplate

Prototypes	ReturnCode generateEnrollmentTemplate(	
	const **ONEVIDEO** &inputVideo,	Input
	std::vector<**PERSONREP**> &enrollTemplates);	Output
Description	This function takes an **ONEVIDEO**, and outputs a vector of **PERSONREP** objects. If the function executes correctly (i.e. returns a ReturnCode::Success exit status), the NIST calling application will store the template. The NIST application will concatenate the templates and pass the result to the enrollment finalization function. If the function gives a non-zero exit status: – If the exit status is ReturnCode::FailParse, NIST will debug, otherwise – the test driver will ignore the output template (the template may have any size including zero) – the event will be counted as a failure to enroll. Such an event means that this person can never be identified correctly. IMPORTANT. NIST's application writes the template to disk. The implementation must not attempt writes to the enrollment directory (nor to other resources). Any data needed during subsequent searches should be included in the template, or created from the templates during the enrollment finalization function.	
Input Parameters	inputVideo	An instance of a Table 36 class.
Output Parameters	enrollTemplates	For each person detected in the **ONEVIDEO**, the function shall identify the person's estimated eye centers for each video frame where the person's eye coordinates can be calculated. The eye coordinates shall be captured in the **PERSONREP**.eyeCoordinates variable, which is a vector of **EYEPAIR** objects. The frame number from the video of where the eye coordinates were detected shall be captured in the **EYEPAIR**.frameNum variable for each pair of eye coordinates. In the event the eye centers cannot be calculated (ie. the person becomes out of sight for a few frames in the video), the SDK shall not store an **EYEPAIR** for those frames.
ReturnCode	Success	Success
	RefuseInput	Elective refusal to process this kind of **ONEVIDEO**
	FailExtract	Involuntary failure to extract features (e.g. could not find face in the input-image)
	FailTempl	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	FailParse	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.6.10. The VideoFinalize Interface

The abstract class VideoFinalize must be implemented by the SDK developer in a class named exactly SdkVideoFinalize. The finalize function in this class takes the name of the top-level directory where enrollment database (EDB) and its manifest have been stored. These are described in section 0. The enrollment directory permissions will be read + write.

	C++ code fragment	Remarks
1.	`class VideoFinalize`	
2.	`{` `public:`	

3.	`virtual ReturnCode finalize(`
	`const string &enrollDir,`
	`const string &edbName,`
	`const string &edbManifest) = 0;`
		This function supports post-enrollment developer-optional book-keeping operations and statistical processing. The function will generally be called in a separate process after all the enrollment processes are complete.

4.	`// Destructor`	
5.	`};`	

3.6.11. Finalize video enrollment

After all templates have been created, the function of Table 46 will be called. This freezes the enrollment data. After this call the enrollment dataset will be forever read-only. This API does not support interleaved enrollment and search phases.

The function allows the implementation to conduct, for example, statistical processing of the feature data, indexing and data re-organization. The function may alter the file structure. It may increase or decrease the size of the stored data. No output is expected from this function, except a return code.

Table 46 – VideoFinalize::finalize

Prototypes	ReturnCode finalize (	
	const string &enrollDir,	Input
	const string &edbName,	Input
	const string &edbManifest);	Input
Description	This function takes the name of the top-level directory where enrollment database (EDB) and its manifest have been stored. These are described in section 0. The enrollment directory permissions will be read + write. The function supports post-enrollment developer-optional book-keeping operations and statistical processing. The function will generally be called in a separate process after all the enrollment processes are complete. This function should be tolerant of being called two or more times. Second and third invocations should probably do nothing.	
Input Parameters	enrollDir	The top-level directory in which enrollment data was placed. This variable allows an implementation to locate any private initialization data it elected to place in the directory.
	edbName	The name of a single file containing concatenated templates, i.e. the EDB of section 0. While the file will have read-write-delete permission, the SDK should only alter the file if it preserves the necessary content, in other files for example. The file may be opened directly. It is not necessary to prepend a directory name.
	edbManifest	The name of a single file containing the EDB manifest of section 0. The file may be opened directly. It is not necessary to prepend a directory name.
Output Parameters	None	
ReturnCode	Success	Success
	FinInputData	Cannot locate the input data - the input files or names seem incorrect.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission, space).
	FinTemplFormat	One or more template files are in an incorrect format.
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.6.12. The VideoFeatureExtraction Interface

The abstract class VideoFeatureExtraction must be implemented by the SDK developer in a class named exactly SdkVideoFeatureExtraction.

	C++ code fragment	Remarks
1.	`class VideoFeatureExtraction`	
2.	`{` `public:`	

3.	`virtual ReturnCode initialize(`
	`const string &configDir,`
	`const string &enrollDir) = 0;`
		Initialize the feature extraction session.

4.	`virtual ReturnCode generateIdTemplate(`
	`const ONEVIDEO &inputVideo,`
	`vector<PERSONREP> &idTemplates) = 0;`
		Generate identification template(s) for the persons detected in the input video. This function takes an **ONEVIDEO** (see 3.6.2) as input and populates a vector of **PERSONREP** (see 3.6.5) with the number of persons detected from the video sequence. The implementation could call vector::push_back to insert into the vector.

5.	`// Destructor`	
6.	`};`	

3.6.13. Video feature extraction initialization

Before one or more **ONEVIDEO**s are sent to the identification feature extraction function, the test harness will call the initialization function below.

Table 47 – VideoFeatureExtraction::initialize

Prototype	ReturnCode initialize(	
	const string &configDir,	Input
	const string &enrollDir);	Input
Description	This function initializes the SDK under test and sets all needed parameters. This function will be called once by the NIST application immediately before any M ≥ 1 calls to generateIdTemplate. The SDK should tolerate execution of P => 1 processes on the same machine each of which can read the configuration directory. This function may be called P times and these may be running simultaneously and in parallel. The implementation has read-only access to its prior enrollment data.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The top-level directory in which enrollment data was placed and then finalized by the implementation. The implementation can parameterize subsequent template production on the basis of the enrolled dataset.
Output Parameters	none	
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission).
	Vendor	Vendor-defined failure

3.6.13.1. Video feature extraction

An **ONEVIDEO** is converted to one or more identification templates using the function below. The result may be stored by NIST, or used immediately. The SDK shall not attempt to store any data.

Table 48 – VideoFeatureExtraction::generateIdTemplate

Prototypes	ReturnCode generateIdTemplate(	
	const **ONEVIDEO** &inputVideo,	Input

	std::vector<**PERSONREP**> &idTemplates);		Output
Description	This function takes an **ONEVIDEO** (see 3.6.2) as input and populates a vector of **PERSONREP** (see 3.6.5) with the number of persons detected from the video sequence. The implementation could call vector::push_back to insert into the vector. If the function executes correctly, it returns a zero exit status. The NIST calling application may commit the template to permanent storage, or may keep it only in memory (the implementation does not need to know). If the function returns a non-zero exit status, the output template will be not be used in subsequent search operations. The function shall not have access to the enrollment data, nor shall it attempt access.		
Input Parameters	InputVideo	An instance of a section 3.6.2 class. Implementations must alter their behavior according to the people detected in the video sequence.	
Output Parameters	IdTemplates	For each person detected in the video, the function shall create a **PERSONREP** (see section 3.6.5) object, populate it with a template and eye coordinates for each frame where eyes were detected, and add it to the vector.	
ReturnCode	Success	Success	
	RefuseInput	Elective refusal to process this kind of **ONEVIDEO**	
	FailExtract	Involuntary failure to extract features (e.g. could not find face in the input-image)	
	FailTempl	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)	
	FailParse	Cannot parse input data (i.e. assertion that input record is non-conformant)	
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.	

3.6.14. The VideoSearch Interface

The abstract class VideoSearch must be implemented by the SDK developer in a class named exactly SdkVideoSearch.

	C++ code fragment	Remarks
1.	`class VideoSearch`	
2.	`{` `public:`	

3.	`virtual ReturnCode initialize(`
	`const string &configDir,`
	`const string &enrollDir) = 0;`
		Initialize the search session.

4.	`virtual ReturnCode identifyVideo(`
	`const PERSONREP &idVideoTemplate,`
	`const uint32_t candListLength,`
	`CANDIDATELIST &candList) = 0;`
		For video-to-video identification
		
		This function searches a template generated from an **ONEVIDEO** against the enrollment set, and outputs a vector containing candListLength objects of Candidates (see section 3.6.7).

5.	`virtual ReturnCode identifyImage(`
	`const PERSONREP &idImageTemplate,`
	`const uint32_t candListLength,`
	`CANDIDATELIST &candList) = 0;`
		For still-to-video identification
		
		This function searches a template generated from a **MULTIFACE** against the enrollment set, and outputs a vector containing candListLength objects of Candidates.

6.	`// Destructor`	
7.	`};`	

3.6.14.1. Video identification initialization

The function below will be called once prior to one or more calls of the searching function of Table 50. The function might set static internal variables so that the enrollment database is available to the subsequent identification searches.

Table 49 – VideoSearch::initialize

Prototype	ReturnCode initialize(	
	const string &configDir,	Input
	const string &enrollDir);	Input
Description	This function reads whatever content is present in the enrollment_directory, for example a manifest placed there by the	

	VideoFinalize::finalize function.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The top-level directory in which enrollment data was placed.
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission).
	Vendor	Vendor-defined failure

3.6.15. Video identification search

The function below compares a proprietary identification template against the enrollment data and returns a candidate list.

Table 50 – VideoSearch::identifyVideo and VideoSearch::identifyImage

Prototype	ReturnCode identifyVideo(	Searches a template generated from a **ONEVIDEO** against the enrollment set (video-to-video)
	const **PERSONREP** &idVideoTemplate,	Input
	const uint32_t candListLength,	Input
	CANDIDATELIST &candList);	Output
	ReturnCode identifyImage(	Searches a template generated from a **MULTIFACE** against the enrollment set (still-to-video)
	const **PERSONREP** &idImageTemplate,	Input
	const uint32_t candListLength,	Input
	CANDIDATELIST &candList);	Output
Description	This function searches an identification template against the enrollment set, and outputs a vector containing candListLength Candidates (see section 3.6.7). Each candidate shall be populated by the implementation and added to candList. Note that candList will be an empty vector when passed into this function. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.	
Input Parameters	idTemplate	A template from generateIdTemplate() - If the value returned by that function was non-zero the contents of idTemplate will not be used and this function (i.e. identifyVideo) will not be called.
	candListLength	The number of candidates the search should return
Output Parameters	candList	A vector containing candListLength objects of Candidates. The datatype is defined in section 3.6.7. Each candidate shall be populated by the implementation and added to this vector. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.
ReturnCode	Success	Success
	IdBadTempl	The input template was defective.
	Vendor	Vendor-defined failure

3.6.16. The ImageEnrollment Interface

The abstract class ImageEnrollment must be implemented by the SDK developer in a class named exactly SdkImageEnrollment.

	C++ code fragment	Remarks
1.	`class ImageEnrollment`	
2.	`{` `public:`	
3.	`virtual ReturnCode getPid(` `string &sdkId, string &email) = 0;`	Return the sdk identifier and email

4.	`virtual ReturnCode initialize(`
	`const string &configDir,`
	`const string &enrollDir,`
	`const uint32_t numPersons,`
	`const uint32_t numImages,`
	`const vector<string> &descriptions) = 0 ;`
		Initialize the enrollment session.

5.	`virtual ReturnCode generateEnrollmentTemplate(`
	`const MULTIFACE &inputFaces,`
	`PERSONREP &outputTemplate) = 0;`
		This function takes a **MULTIFACE** (see 2.3.3) as input and outputs a proprietary template represented by a **PERSONREP** (see 3.6.5).
		
		For each input image in the **MULTIFACE**, the function shall return the estimated eye centers by setting **PERSONREP**.eyeCoordinates.

6.	`// Destructor`	
7.	`};`	

3.6.17. Implementation identifier

Table 51 – ImageEnrollment::getPid

Prototype	ReturnCode getPid(	
	string &sdkId,	A developer-assigned ID. This shall be different for each submitted SDK.
	string &email);	Output
Description	This function retrieves a point-of-contact email address from the implementation under test.	
Output Parameters	sdkId	4-character version ID code as hexadecimal integer. This will be used to identify the SDK in the results reports. This value should be changed every time an SDK is submitted to NIST. The value is developer assigned - format is not regulated by NIST. EXAMPLE: "011A". The value cannot be the empty string.
	email	Point of contact email address. The value cannot be the empty string.
ReturnCode	Success	Success
	Vendor	Vendor-defined failure

3.6.17.1. Initialization of the image enrollment session

Before any enrollment feature extraction calls are made, the NIST test harness will call the initialization below for video-to-still.

Table 52 – ImageEnrollment::initialize

Prototype	ReturnCode initialize(	
	const string &configDir,	Input
	const string &enrollDir,	Input
	const uint32_t numPersons,	Input
	const uint32_t numImages,	Input
	const std::vector<string> &descriptions);	Input
Description	This function initializes the SDK under test and sets all needed parameters. This function will be called N=1 times by the NIST application immediately before any M ≥ 1 calls to generateEnrollmentTemplate. The SDK should tolerate execution of P > 1 processes on the same machine each of which may be reading and writing to the enrollment directory. This function may be called P times and these may be running simultaneously and in parallel.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The directory will be initially empty, but may have been initialized and populated by separate invocations of the enrollment process. When this function is called, the SDK may populate

		this folder in any manner it sees fit. Permissions will be read-write-delete.
	numPersons	The number of persons who will be enrolled.
	numImages	The total number of images that will be enrolled, summed over all identities.
	descriptions	A lexicon of labels one of which will be assigned to each enrollment image. EXAMPLE: The descriptions could be {"mugshot", "visa"}. NOTE: The identification search images may or may not be labeled. An identification image may carry a label not in this set of labels. The number of items stored in the vector is accessible via the vector::size() function.
Output Parameters	none	
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission, space).
	InitNumData	The SDK cannot support the number of videos.
	InitBadDesc	The descriptions are unexpected, or unusable.
	Vendor	Vendor-defined failure

3.6.17.2. Image enrollment

A **MULTIFACE** (see Table 12) is converted to a single enrollment template using the function below.

Table 53 – ImageEnrollment::generateEnrollmentTemplate

Prototypes	ReturnCode generateEnrollmentTemplate(	
	const **MULTIFACE** &inputFaces,	Input
	PERSONREP &outputTemplate);	Output
Description	This function takes a **MULTIFACE**, and outputs a proprietary template in the form of a **PERSONREP** object. If the function executes correctly (i.e. returns a ReturnCode::Success exit status), the NIST calling application will store the template. The NIST application will concatenate the templates and pass the result to the enrollment finalization function. If the function gives a non-zero exit status: – If the exit status is ReturnCode::FailParse, NIST will debug, otherwise – the test driver will ignore the output template (the template may have any size including zero) – the event will be counted as a failure to enroll. Such an event means that this person can never be identified correctly. IMPORTANT. NIST's application writes the template to disk. The implementation must not attempt writes to the enrollment directory (nor to other resources). Any data needed during subsequent searches should be included in the template, or created from the templates during the enrollment finalization function.	
Input Parameters	inputFaces	An instance of a Table 12 structure.
Output Parameters	outputTemplate	An instance of a section 3.6.5 class, which stores proprietary template data and eye coordinates. The function shall identify the person's estimated eye centers for each image in the **MULTIFACE**. The eye coordinates shall be captured in the **PERSONREP**.eyeCoordinates variable, which is a vector of **EYEPAIR** objects. In the event the eye centers cannot be calculated, the SDK shall store an **EYEPAIR** and set **EYEPAIR**.isSet to false to indicate there was a failure in generating eye coordinates. In other words, for N images in the **MULTIFACE**.
ReturnCode	Success	Success
	RefuseInput	Elective refusal to process this kind of **ONEVIDEO**
	FailExtract	Involuntary failure to extract features (e.g. could not find face in the input-image)
	FailTempl	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	FailParse	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.6.18. The ImageFinalize Interface

The abstract class ImageFinalize must be implemented by the SDK developer in a class named exactly SdkImageFinalize. The finalize function in this class takes the name of the top-level directory where enrollment database (EDB) and its manifest have been stored. These are described in section 0. The enrollment directory permissions will be read + write.

	C++ code fragment	Remarks
1.	`class ImageFinalize`	
2.	`{` `public:`	

3.	`virtual ReturnCode finalize(`
	`const string &enrollDir,`
	`const string &edbName,`
	`const string &edbManifest) = 0;`
		This function supports post-enrollment developer-optional book-keeping operations and statistical processing. The function will generally be called in a separate process after all the enrollment processes are complete.

4.	`// Destructor`	
5.	`};`	

3.6.19. Finalize image enrollment

After all templates have been created, the function of Table 54 will be called. This freezes the enrollment data. After this call the enrollment dataset will be forever read-only. This API does not support interleaved enrollment and search phases.

The function allows the implementation to conduct, for example, statistical processing of the feature data, indexing and data re-organization. The function may alter the file structure. It may increase or decrease the size of the stored data. No output is expected from this function, except a return code.

Table 54 – ImageFinalize::finalize

Prototypes	ReturnCode finalize(	
	const string &enrollDir,	Input
	const string &edbName,	Input
	const string &edbManifest);	Input
Description	This function takes the name of the top-level directory where enrollment database (EDB) and its manifest have been stored. These are described in section 0. The enrollment directory permissions will be read + write. The function supports post-enrollment developer-optional book-keeping operations and statistical processing. The function will generally be called in a separate process after all the enrollment processes are complete. This function should be tolerant of being called two or more times. Second and third invocations should probably do nothing.	
Input Parameters	enrollDir	The top-level directory in which enrollment data was placed. This variable allows an implementation to locate any private initialization data it elected to place in the directory.
	edbName	The name of a single file containing concatenated templates, i.e. the EDB of section 0. While the file will have read-write-delete permission, the SDK should only alter the file if it preserves the necessary content, in other files for example. The file may be opened directly. It is not necessary to prepend a directory name.
	edbManifest	The name of a single file containing the EDB manifest of section 0. The file may be opened directly. It is not necessary to prepend a directory name.
Output Parameters	None	
ReturnCode	Success	Success
	FinInputData	Cannot locate the input data - the input files or names seem incorrect.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission, space).
	FinTemplFormat	One or more template files are in an incorrect format.
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.6.20. The ImageFeatureExtraction Interface

The abstract class ImageFeatureExtraction must be implemented by the SDK developer in a class named exactly SdkImageFeatureExtraction.

	C++ code fragment	Remarks
1.	`class ImageFeatureExtraction`	
2.	`{` `public:`	

3.	`virtual ReturnCode initialize(`
	`const string &configDir,`
	`const string &enrollDir) = 0;`
		Initialize the feature extraction session.

4.	`virtual ReturnCode generateIdTemplate(`
	`const MULTIFACE &inputFaces,`
	`PERSONREP &outputTemplate) = 0;`
		This function takes a **MULTIFACE** (see 2.3.3) as input and outputs a proprietary template represented by a **PERSONREP** (see 3.6.5).
		
		For each input image in the **MULTIFACE**, the function shall return the estimated eye centers by setting **PERSONREP**.eyeCoordinates.

5.	`// Destructor`	
6.	`};`	

3.6.20.1. Image feature extraction initialization

Before one or more **MULTIFACE**s are sent to the identification feature extraction function, the test harness will call the initialization function below.

Table 55 – ImageFeatureExtraction::initialize

Prototype	ReturnCode initialize(	
	const string &configDir,	Input
	const string &enrollDir);	Input
Description	This function initializes the SDK under test and sets all needed parameters. This function will be called once by the NIST application immediately before M ≥ 1 calls to generateIdTemplate. The SDK should tolerate execution of P ≥ 1 processes on the same machine each of which can read the configuration directory. This function may be called P times and these may be running simultaneously and in parallel. The implementation has read-only access to its prior enrollment data.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The top-level directory in which enrollment data was placed and then finalized by the implementation. The implementation can parameterize subsequent template production on the basis of the enrolled dataset.
Output Parameters	none	
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission).
	Vendor	Vendor-defined failure

3.6.20.2. Image feature extraction

A **MULTIFACE** is converted to one identification template using the function below. The result may be stored by NIST, or used immediately. The SDK shall not attempt to store any data.

Table 56 – ImageFeatureExtraction::generateIdTemplate

Prototypes	ReturnCode generateIdTemplate(	
	const **MULTIFACE** &inputFaces,	Input

	PERSONREP &outputTemplate);	Output
Description	This function takes a **MULTIFACE** (see 2.3.3) as input and populates a **PERSONREP** (see 3.6.5) with a proprietary template and eye coordinates. If the function executes correctly, it returns a zero exit status. The NIST calling application may commit the template to permanent storage, or may keep it only in memory (the developer implementation does not need to know). If the function returns a non-zero exit status, the output template will be not be used in subsequent search operations. The function shall not have access to the enrollment data, nor shall it attempt access.	
Input Parameters	inputFaces	An instance of a Table 12 structure.
Output Parameters	outputTemplate	An instance of a section 3.6.5 class, which stores proprietary template data and eye coordinates. The function shall identify the person's estimated eye centers for each image in the **MULTIFACE**. The eye coordinates shall be captured in the **PERSONREP**.eyeCoordinates variable, which is a vector of **EYEPAIR** objects. In the event the eye centers cannot be calculated, the SDK shall store an **EYEPAIR** and set **EYEPAIR**.isSet to false to indicate there was a failure in generating eye coordinates. In other words, for N images in the **MULTIFACE**.
ReturnCode	Success	Success
	RefuseInput	Elective refusal to process this kind of **ONEVIDEO**
	FailExtract	Involuntary failure to extract features (e.g. could not find face in the input-image)
	FailTempl	Elective refusal to produce a template (e.g. insufficient pixels between the eyes)
	FailParse	Cannot parse input data (i.e. assertion that input record is non-conformant)
	Vendor	Vendor-defined failure. Failure codes must be documented and communicated to NIST with the submission of the implementation under test.

3.6.21. The ImageSearch Interface

The abstract class ImageSearch must be implemented by the SDK developer in a class named exactly SdkImageSearch.

	C++ code fragment	Remarks
1.	`class VideoFeatureExtraction`	
2.	`{` `public:`	

3.	`virtual ReturnCode initialize(`
	`const string &configDir,`
	`const string &enrollDir) = 0;`
		Initialize the search session.

4.	`virtual ReturnCode identifyVideo(`
	`const PERSONREP &idTemplate,`
	`const uint32_t candListLength,`
	`CANDIDATELIST &candList) = 0;`
		For video-to-still identification
		
		This function searches a template generated from an **ONEVIDEO** against the enrollment set, and outputs a vector containing candListLength objects of Candidates (see section 3.6.7). Each candidate shall be populated by the implementation and added to candList. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.

5.	`// Destructor`	
6.	`};`	

3.6.21.1. Image identification initialization

The function below will be called once prior to one or more calls of the searching function of Table 58. The function might set static internal variables so that the enrollment database is available to the subsequent identification searches.

Table 57 – ImageSearch::initialize

Prototype	ReturnCode initialize(	
	const string &configDir,	Input
	const string &enrollDir);	Input

Description	This function reads whatever content is present in the enrollment_directory, for example a manifest placed there by the ImageFinalize::finalize function.	
Input Parameters	configDir	A read-only directory containing any developer-supplied configuration parameters or run-time data files.
	enrollDir	The top-level directory in which enrollment data was placed.
ReturnCode	Success	Success
	MissingConfig	The configuration data is missing, unreadable, or in an unexpected format.
	EnrollDirFailed	An operation on the enrollment directory failed (e.g. permission).
	Vendor	Vendor-defined failure

3.6.22. Image identification search

The function below performs a video-to-still identification and compares a proprietary identification template generated from a video against the enrollment data and returns a candidate list.

Table 58 – ImageSearch::identifyVideo

Prototype	ReturnCode identifyVideo(	Searches a template generated from a **ONEVIDEO** against the enrollment set (video-to-still)
	const **PERSONREP** &idVideoTemplate,	Input
	const uint32_t candListLength,	Input
	CANDIDATELIST &candList);	Output

Description	This function searches an identification template against the enrollment set, and outputs a vector containing candListLength objects of Candidates (see section 3.6.7). Each candidate shall be populated by the implementation and added to candList. Note that candList will be an empty vector when passed into this function. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.	
Input Parameters	idTemplate	A template from VideoFeatureExtraction::generateIdTemplate() - If the value returned by that function was non-zero the contents of idTemplate will not be used and this function (i.e. identifyVideo) will not be called.
	candListLength	The number of candidates the search should return
Output Parameters	candList	A vector containing candListLength objects of Candidates. The datatype is defined in section 3.6.7. Each candidate shall be populated by the implementation and added to this vector. The candidates shall appear in descending order of similarity score - i.e. most similar entries appear first.
ReturnCode	Success	Success
	IdBadTempl	The input template was defective.
	Vendor	Vendor-defined failure

4. References

FRVT 2002	Face Recognition Vendor Test 2002: Evaluation Report, NIST Interagency Report 6965, P. Jonathon Phillips, Patrick Grother, Ross J. Micheals, Duane M. Blackburn, Elham Tabassi, Mike Bone
FRVT 2002b	Face Recognition Vendor Test 2002: Supplemental Report, NIST Interagency Report 7083, Patrick Grother
FRVT 2006	P. Jonathon Phillips, W. Todd Scruggs, Alice J. O'Toole, Patrick J. Flynn, Kevin W. Bowyer, Cathy L. Schott, and Matthew Sharpe. "FRVT 2006 and ICE 2006 Large-Scale Results." NISTIR 7408, March 2007.
AN27	NIST Special Publication 500-271: American National Standard for Information Systems — Data Format for the Interchange of Fingerprint, Facial, & Other Biometric Information – Part 1. (ANSI/NIST ITL 1-2007). Approved April 20, 2007.
IREX III	P. Grother, G.W. Quinn, J. Matey, M. Ngan, W. Salamon, G. Fiumara, C. Watson, Iris Exchange III, Performance of Iris Identification Algorithms, NIST Interagency Report 7836, Released April 9, 2012. http://iris.nist.gov/irex
MBE	P. Grother, G .W. Quinn, and P. J. Phillips, Multiple-Biometric Evaluation (MBE) 2010, Report on the Evaluation of 2D Still Image Face Recognition Algorithms, NIST Interagency Report 7709, Released June 22, 2010. Revised August 23, 2010. http://face.nist.gov/mbe
MINEX	P. Grother et al., Performance and Interoperability of the INCITS 378 Template, NIST IR 7296 http://fingerprint.nist.gov/minex04/minex_report.pdf
MOC	P. Grother and W. Salamon, MINEX II - An Assessment of ISO/IEC 7816 Card-Based Match-on-Card Capabilities http://fingerprint.nist.gov/minex/minexII/NIST_MOC_ISO_CC_interop_test_plan_1102.pdf
PERFSTD INTEROP	ISO/IEC 19795-4 — Biometric Performance Testing and Reporting — Part 4: Interoperability Performance Testing. Posted as document 37N2370. The standard was published in 2007. It can be purchased from ANSI at http://webstore.ansi.org/.
ISO STD05	ISO/IEC 19794-5:2005 — Information technology — Biometric data interchange formats — Part 5: Face image data. The standard was published in 2005, and can be purchased from ANSI at http://webstore.ansi.org/ Multipart standard of "Biometric data interchange formats". This standard was published in 2005. It was amended twice to include guidance to photographers, and then to include 3D information. Two corrigenda were published. All these changes and new material is currently being incorporated in revision of the standard. Publication is likely in early 2011. The documentary history is as follows. ISO/IEC 19794-5: Information technology — Biometric data interchange formats — Part 5:Face image data. First edition: 2005-06-15. International Standard ISO/IEC 19794-5:2005 Technical Corrigendum 1: Published 2008-07-01 International Standard ISO/IEC 19794-5:2005 Technical Corrigendum 2: Published 2008-07-01 Information technology — Biometric data interchange formats — Part 5: Face image data AMENDMENT 1: Conditions for taking photographs for face image data. Published 2007-12-15 Information technology — Biometric data interchange formats — Part 5: Face image data AMENDMENT 2: Three dimensional image data. JTC 1/SC37/N3303. FCD text of the second edition. Contact pgrother AT nist DOT gov for more information.

Annex A
Submission of Implementations to the FRVT 2012

A.1 Submission of implementations to NIST

NIST requires that all software, data and configuration files submitted by the participants be signed and encrypted. Signing is done with the participant's private key, and encryption is done with the NIST public key. The detailed commands for signing and encrypting are given here: http://www.nist.gov/itl/iad/ig/encrypt.cfm

NIST will validate all submitted materials using the participant's public key, and the authenticity of that key will be verified using the key fingerprint. This fingerprint must be submitted to NIST by writing it on the signed participation agreement.

By encrypting the submissions, we ensure privacy; by signing the submission, we ensure authenticity (the software actually belongs to the submitter). NIST will reject any submission that is not signed and encrypted. NIST accepts no responsibility for anything that is transmitted to NIST that is not signed and encrypted with the NIST public key.

A.2 How to participate

Those wishing to participate in FRVT 2012 testing must do all of the following, on the schedule listed on Page 1.

— IMPORTANT: Follow the instructions for cryptographic protection of your SDK and data here. http://www.nist.gov/itl/iad/ig/encrypt.cfm

— Send a signed and fully completed copy of the *Application to Participate in the Face Recognition Vendor Test (FRVT) 2012*. This is available at http://www.nist.gov/itl/iad/ig/frvt-2012.cfm. This must identify, and include signatures from, the Responsible Parties as defined in the application. The properly signed FRVT 2012 Application to Participate shall be sent to NIST as a PDF.

— Provide an SDK (Software Development Kit) library which complies with the API (Application Programmer Interface) specified in this document.

- Encrypted data and SDKs below 20MB can be emailed to NIST at frvt2012@nist.gov
- Encrypted data and SDKS above 20MB shall be
  - Made available as a file.zip.gpg or file.zip.asc download from a generic webserver[16], or:
  - Mailed as a file.zip.gpg or file.zip.asc on CD / DVD to NIST at this address:

FRVT 2012 Test Liaison (A203) 100 Bureau Drive A203/Tech225/Stop 8940 NIST Gaithersburg, MD 20899-8940 USA	In cases where a courier needs a phone number, please use NIST shipping and handling on: 301 -- 975 -- 6296.

A.3 Implementation validation

Registered Participants will be provided with a small validation dataset and test program available on the website http://www.nist.gov/itl/iad/ig/frvt-2012.cfm shortly after the final evaluation plan is released.

The validation test programs shall be compiled by the provider. The output of these programs shall be submitted to NIST.

Prior to submission of the SDK and validation data, the Participant must verify that their software executes on the validation images, and produces correct similarity scores and templates.

Software submitted shall implement the FRVT 2012 API Specification as detailed in the body of this document.

[16] NIST will not register, or establish any kind of membership, on the provided website.

Upon receipt of the SDK and validation output, NIST will attempt to reproduce the same output by executing the SDK on the validation imagery, using a NIST computer. In the event of disagreement in the output, or other difficulties, the Participant will be notified.